Addition of ElGamal Plaintexts

Markus Jakobsson[1] and Ari Juels[2]

[1] Information Sciences Research Center
Bell Labs
Murray Hill, New Jersey 07974
`www.bell-labs.com/user/markusj/`
[2] RSA Laboratories
RSA Security Inc.
Bedford, MA 01730, USA
`ajuels@rsasecurity.com`

Abstract. We introduce an efficient method for performing computation on encrypted data, allowing addition of ElGamal encrypted plaintexts. We demonstrate a solution that is robust and leaks no information to a minority of colluding cheaters. Our focus is on a three-player solution, but we also consider generalization to a larger number of players. The amount of work is exponential in the number of players, but reasonable for small sets.

Keywords: addition, directed decryption, ElGamal, fast-track, repetition robustness

1 Introduction

For a number of public-key encryption schemes, it possible to compose plaintexts multiplicatively through simple manipulation of their corresponding ciphertexts, without knowledge of an associated decryption key. For ElGamal encryption, this is achieved by component-wise multiplication of individual ciphertexts. If the ciphertexts have been encrypted under the same public key, the result is an ElGamal encryption of the product of the plaintexts. It is known that there can not exist a *non-interactive* method for computing the *sum* of plaintexts in this setting given only the ciphertexts. If so, then it would be possible to break the Decision Diffie-Hellman Assumption.[1] Furthermore, even allowing interaction, and working in a setting where the decryption key is held distributively, it is not known how to compute the sum of plaintexts without invocation of costly general secure multiparty computation methods. In this paper, we propose the first efficient solution for performing the operation of distributed ElGamal plaintext

[1] The ability to perform non-interactive plaintext addition implies the ability to build a comparitor that determines whether a ciphertext corresponds to a particular plaintext. This would contradict the semantic security of the cipher, which is known to hold if the DDH assumption holds. The comparitor would add an encryption of the additive inverse of the assumed plaintext, and determine whether the result of the addition is an encryption of the value zero, which is the only recognizable ciphertext.

addition. However, we note that our solution is only efficient for small number of participants, as its asymptotic costs are exponential in the number of players. We hope that improvements to our techniques will be able to remove this restriction.

Our goal is therefore to solve the following problem: Two ElGamal ciphertexts, both encrypted using a public key y, are given to a set of n players, among whom the corresponding secret key x is shared using a standard (k, n)-threshold secret sharing scheme. The players wish to generate an encryption of the sum of the encrypted plaintexts. We want the computation to be done in a robust manner, i.e., such that cheating will be detected and traced with an overwhelming probability. At the same time, it must be performed without revealing any information to a minority of dishonest and colluding players. In the main body of the paper, we focus on a $(2, 3)$-threshold solution, but this solution generalizes rather straightforwardly to more players. (However, as mentioned, the solution is only practical for small sets.)

We want our multiplicative group to be closed under addition. Working in the field $F[2^t]$ gives us a close approximation of this. In order not to leak the residuosity of results, we additionally require the size of the multiplicative group to be prime. We review the related security aspects of this modification.

Our method is based on iterated additive resharing of secrets and on a handful of scheduling tricks. Since an important goal is for the protocol to be computationally efficient, we avoid traditional zero-knowledge based robustness methods, and employ instead the so-called *repetition robustness* method [9]. We expand on the use of this method by offering a new type of setting in which it is beneficial. We also employ the ideas of *fast-track computation*, optimizing the scheme for the case where no players attempt to corrupt the computation. (See [7] for the introduction of the term and, e.g., [5] for an early introduction to the idea.)

Apart from addressing a long open problem, we introduce methods for efficient computation that may be of independent interest. For example, our result might serve as a possible alternative basis for general secure multiparty computation. It is interesting to note that this would give us a solution where one pays for *addition*, while *multiplications* are (almost) for free. This is in contrast to the usual case for general secure multiparty computation, where addition is free while multiplication is relatively costly. It appears that for some types of computations, our approach would lower the cost of the function evaluation.

Our building block may also find applications in more specific multi-party settings, such as certification schemes, payment schemes, and election schemes. This may be both to allow new functionality, and to lower computational costs by allowing for alternative ways of arriving at a given result. It is interesting to note that it sometimes may lower computational costs to perform computation on encrypted data, compared to traditional secret-sharing methods in which each party needs to prove that he performed the correct computation.

We therefore see our work as an interesting step in a direction that may provide new types of solutions for multi-party computation. We do not, however, view the work presented here to be conclusive in any manner. Instead, we want to bring the attention to several remaining open problems. Most notable among these are the questions of whether there exists an efficient and secure two-player solution, and whether an efficient solution can be developed for large sets of players, ours being efficient only for moderate sized sets.

Outline. We begin with a review of related work (Section 2). We then introduce our building blocks and design methods (Section 3), and present a description of a non-robust solution (Section 4). In Section 5, we show how to incorporate robustness into that solution. We prove that our solution satisfies our stated requirements in Appendix A.

2 Related Work

We introduce the notion of one-splitting. This is superficially related to standard secret-sharing methods [17], and in particular to zero-sharing (see, e.g., [11]). The latter involves superimposition of several polynomials, each encoding the value zero, in order to create an unknown polynomial that likewise encodes a zero. While the technical differences are substantial, the principles are related. Our methods involve composition of multiple ciphertexts, resulting in a set of new ciphertexts such that the sum of their plaintexts equals the value one.

We use the notion of repetition robustness, introduced in [9] and later also employed in [10], to make our protocol robust without excessive use of zero-knowledge proofs. If zero-knowledge proofs were to be used instead, it would appear necessary to invoke computationally costly cut-and-choose methods. (While there is nothing inherent in the setting that requires a cut-and-choose approach, we are not aware of any other kind of zero-knowledge method that can be used to perform the required proof.)

The version of repetition robustness outlined in [9, 10] relies on the use of permutation, and potentially requires a large number of repetitions in order to render the success probability of an attacker sufficiently small. Our flavor of the principle, however, does not involve permutation, and only requires two runs of the protocol to ensure a negligible probability of success for an attacker. This is because our protocol by its very nature destroys the homomorphic properties exploitable by an attacker in [9, 10].

Our solution is based on the principles of fast-track computation [7, 5]. The guiding aim here is to streamline the efficiency of the protocol for the case in which no player attempts to corrupt the output, in the hope that this is the most common case. This translates to inexpensive methods for detection of errors, followed by a conditional execution of potentially expensive methods for determining which player misbehaved.

We use standard methods for proving correctness of exponentiation. These are related to verification of undeniable signatures [2, 1] and to discrete log based signatures. We refer to [8] for methods relating to the latter approach.

All computation takes place in $F[2^t]$, where $q = 2^t - 1$ is prime. This is done to avoid leaks of information relating to the Jacobi symbol of the result of the computation, and to achieve "approximate closure" of the multiplicative group under addition. It should be noted that performing the computation in this structure allows for more efficient attacks than for the standard parameter choices, as shown by Coppersmith [3]. Coppersmith's result improves the asymptotic running time of the special number field sieve. Implementations [12] suggest that if we perform all arithmetic in a field $F[2^{2203}-1]$ (corresponding to the smallest Mersenne prime of the approximate size we want), instead of the standard choice of 1024 bit moduli, the computational hardness of computing discrete logs would be maintained. We note that the speed of multiplication in this structure is about a quarter of the speed compared to $F[p]$ for $|p| = 1024$ if software is used, and about half if special-purpose hardware is employed.

3 Building Blocks and Design Methods

We employ the standard cryptographic model in which players are modeled by polynomial-time Turing Machines. We make the common assumption that a minority of players may be dishonest and may collude. A further requirement is the existence of an authenticated broadcast channel among players. We first review standard building blocks presented in the literature, and then introduce some building blocks peculiar to the protocols in this paper.

3.1 Standard building blocks

Group structure. The ElGamal cryptosystem, which we will use, may operate over any of a number of choices of group in which the discrete log problem is infeasible (See [13], for a discussion and list of some proposed group choices.) We let g denote a generator of $F^*[2^t]$, where $p = 2^t - 1$ is prime. We note that the additive and multiplicative groups overlap on all elements but 0.

Secret sharing. We assume that the private decryption key x is shared among players using (k, n)-threshold secret sharing [17], and denote by x_i the secret key share of x held by player i. Here, $x \equiv \sum_{i \in \mathcal{S}} x_i \lambda_{\mathcal{S}i}$, where $\mathcal{S}$ is a set of k players, and $\lambda_{\mathcal{S}i}$ is the Lagrange coefficient for the set $\mathcal{S}$ and player i. It is possible to generate and distribute x using any of a number of well studied protocols. See, e.g., [6] for a brief overview and some caveats.

ElGamal encryption. In the ElGamal cryptosystem, the private key is an integer x selected uniformly at random from Z_p. The corresponding public key is $y = g^x$. In order to encrypt a value $m \in F^*[2^t]$ under public key y, we pick a random element $\alpha \in_u Z_p$, and compute the ciphertext as $(a, b) = (y^\alpha m, g^\alpha)$. (We note that we allow a more "liberal" choice of message encodings than for standard ElGamal encryption, since every element but zero is in the multiplicative group.)

Standard and directed decryption. The plaintext message can be computed from a ciphertext (a, b) by computing a/b^x, where x is the secret decryption key. Note that this trivially allows distribution, where each player, holding an additive share x_i of x, computes and publishes $B_i = b^{x_i}$, from which $B = b^x$ can easily be constructed. We obtain a *directed decryption* for a player i by having all players but i publish their shares of B, after which player i locally computes B_i, then B, and finally the plaintext $m = a/B$.

Multiplication and division of ciphertexts. Let $E(m_1) = (a_1, b_1)$ and $E(m_2) = (a_2, b_2)$ be two ciphertexts, corresponding to plaintexts m_1 and m_2. We say that $E(m_3) = E(m_1)E(m_2) = (a_1a_2, b_1b_2)$ is the *product* of $E(m_1)$ and $E(m_2)$, since its plaintext $m_3 = m_1m_2$. Similarly, we say that the *quotient* of two ciphertexts is $E(m_3) = E(m_1)/E(m_2) = (a_1{a_2}^{-1}, b_1{b_2}^{-1})$, where the resulting ciphertext corresponds to the plaintext $m_3 = m_1{m_2}^{-1}$.

Distributed blinding. For k players to blind a ciphertext $E(m)$, each player i selects a random number $r_i \in F^*[2^t]$ and computes and publishes $E(r_i)$. Then, the players compute $E(\overline{m}) = E(m)\prod_{i=1}^{k} E(r_i)$, for which we have $\overline{m} = m\prod_{i=1}^{k} r_i$. The players unblind $E(\overline{m})$ by computing $E(\overline{m})/(\prod_{i=1}^{k} E(r_i))$, where $E(r_i)$ are the individual blinding factors applied in the blinding step.

Plaintext equality test. This is a distributed protocol for determining whether the plaintexts corresponding to two ElGamal ciphertexts are equal. Given two ciphertexts, E_1 and E_2, we compute $(a, b) = E_1/E_2$. Using, e.g., the techniques described in [8], we then determine in a distributed fashion whether $log_y a = log_g b$. If this equality holds, then the two plaintexts are determined to be equal.

3.2 Special building blocks

We now introduce some building blocks peculiar to our protocols in this paper.

One-splitting. Two parties can compute a one-splitting, i.e., a set of ciphertexts for which the plaintext sum is congruent to 1, using the following approach.

1. The first player selects a random value $w_1 \in_u F[2^t]\ \{0, 1\}$, and computes $\overline{w}_1 \equiv 1 - w_1$. He encrypts these two plaintexts, resulting in the ciphertexts E_1 and $\overline{E}_1$. We call this portion of the one-splitting the "root".
2. The second player selects two random numbers, $w_{2i} \in_u F[2^t]\{0, 1\}$, $1 \leq i \leq 2$, and computes $\overline{w}_{2i} \equiv 1 - w_{2j}$. He encrypts these four plaintexts, giving him E_{21}, $\overline{E}_{21}$, E_{22} and $\overline{E}_{22}$. We call this portion of the one-splitting the "leaves".
3. Both parties commit to their ciphertexts, and then decommit and compute the new ciphertext quadruple $(\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3, \mathcal{E}_4) = (E_1E_{21}, E_1\overline{E}_{21}, \overline{E}_1E_{22}, \overline{E}_1\overline{E}_{22})$. These constitute a one-splitting: It is easy to see that $\sum_{i=1}^{4} \mathcal{E}_i \equiv 1$.

Remark: The one-splitting protocol generalizes straightforwardly to any number of players, but incurs costs exponential in the number of players. It is therefore only suited to small numbers of players. In this paper, we consider a setting in which only two (out of three) players engage in the one-splitting.

Blinded round-robin addition. Let (A, B, C) correspond to three participating players, and let $(M_A, M_B, M_C) \in (F[2^t])^3$ correspond to their respective private inputs. In this protocol, a player "sends" a message m to another player by publishing an encryption $E(m)$, which all players then directively decrypt for the target player. The effect of this procedure is to establish the ciphertext $E(m)$ as a commitment to the transmitted message. This commitment can later be used to trace cheaters if necessary. The blinded round-robin addition protocol is now as follows.

1. A selects $\Delta \in_u F[2^t]$, and then sends $S_1 = M_A + \Delta$ to B and Δ to C.
2. B computes $S_2 = S_1 + M_B$ and sends it to C.
3. C computes $S_3 = S_2 + M_C - \Delta$, and publishes an ElGamal encryption $E(S_3)$ under public key y.

Remark 1: We note that the above protocol will fail to hide the result if $S_3 = 0$. This only happens with a negligible probability for independent and uniformly distributed inputs. For inputs of "dangerous" distributions, we need to split each input value into two portions before performing the addition. This will be described later in the section.

Remark 2: The addition protocol can be extended to $k > 3$ players by having each of the k players compute a (k, k)-threshold sharing of her value. Each player then distributes the pieces of her sharing among all k players. Then, in a round-robin addition, the final player obtains the sum of all shares, for which she outputs the corresponding ciphertext.

Repetition robustness. While standard zero-knowledge based methods can be employed to achieve robustness, the cost for doing so would be substantial. We show how to use a recently introduced method, so-called *repetition robustness* [9], to obtain robustness at low cost. This method works by performing portions of the computation twice, using different random strings for each invocation, and comparing the resulting outputs. We repeat a portion relating to the one-splitting once, and a portion relating to the addition of partial results once, giving us a robust result with a cost less than three times that of the non-robust version of the protocol.

Scheduling tricks. In the primitives we develop, different relations are learned by the different players, and it becomes of vital importance to schedule carefully what player performs which tasks. This is to prevent any player from ending up with a fully determined set of equations and thereby learning information about the plaintexts. The resulting scheduling techniques are remotely related to standard blinding methods. The intuition behind our scheduling methods is as follows: In the different building blocks we have presented, it can be seen that the different parties learn different amounts or relations. For example, the third player in the blinded round-robin addition protocol we presented learns the product of the blinding factor and the message, whereas the other players do not learn this piece of information. In order to ascertain that no player learns any function of the secret information, it is important to schedule the execution of the different building blocks in a manner that does not allow any one participant (or more generally, any set controlled by the adversary) to collect enough relations that he can compute any non-trivial function of secret information. (We will solidify this in the appendix by showing how each party can produce a simulation that is coherent with his view and any set of secret inputs.)

Avoiding zero. ElGamal encryption has the property that the zero plaintext cannot be encrypted, but must be avoided or otherwise encoded, as its corresponding ciphertext is distinguishable from ciphertexts of other plaintexts. Depending on the use of our proposed scheme, and depending on its input, a related problem may be that the whereas no inputs are encryptions of a zero, an intermediary value or an output may still be. In order to avoid this problem, one can represent every item of the computation as a pair of ciphertexts, such that their plaintexts, if added, correspond to the value to be manipulated. We note that it is easy, given our methods, to produce such a "pair representation" of each already encrypted input; this is done plainly by selecting a random ciphertext from the correct distribution, and subtracting this from the initial ciphertext. The result, along with the random ciphertext, is a pair whose plaintext sum corresponds to the plaintext of the original ciphertext. This can be done to all values, after which the desired computation is performed on the ciphertext pairs instead of on the original ciphertexts. (Note that addition can simply be done element-wise, whereas multiplication becomes more laborious.) In our basic solution, we do not consider these issues.

4 A Non-Robust Solution (Protocol $\mathcal{P}_1$)

Using the building blocks introduced in the previous section, we now present a preliminary solution for addition of plaintexts. This solution is correct and complete and implements privacy, but is not robust. Our solution involves three players, two that are active (i.e., are involved in choosing random values) and one that is passive (i.e., only involved in adding values given to him.)

Let $E(m_1)$ and $E(m_2)$ be the input ciphertexts, and $E(m_3)$ the output ciphertext. We assume that the players share the secret decryption key x. Since our protocol is only secure against dishonest minorities, we will use a threshold scheme that reflects the same trust setting. In the three-player setting we consider here, that means that a $(2, 3)$-threshold scheme is employed. Call the following protocol $\mathcal{P}_1$:

1. The two active parties compute a blinding factor $E(r)$ using the methods of *distributed blinding*.
2. The two active parties compute two independent one-splittings. Call these $(\mathcal{E}_{11}, \mathcal{E}_{12}, \mathcal{E}_{13}, \mathcal{E}_{14})$ and $(\mathcal{E}_{21}, \mathcal{E}_{22}, \mathcal{E}_{23}, \mathcal{E}_{24})$. We let the first player set the root of the first one-splitting, and the leaves of the second.

3. In this step we perform the first robustness check; we will elaborate on this in the next section.
4. Let $\mu_{j\kappa} = \mathcal{E}_{j\kappa} E(r) E(m_j)$ for $1 \leq j \leq 2$. The two parties use the methods of *directed decryption* to decrypt the resulting ciphertexts $\mu_{j\kappa}$, giving the first active player the plaintexts of the ciphertexts with $\kappa = 1$, and the second active player the plaintexts of those with $\kappa = 3$. The passive player gets the remaining plaintexts, i.e., those with $\kappa = 2$ and $\kappa = 4$.
5. Each player computes the sum of the above plaintexts. All parties then add these using the blinded round-robin addition protocol. The scheduling order here is $(2, 3, 1)$, i.e., the second active player begins and the first active player finishes, sandwiching the passive player. We denote the result of this step by $E(M)$.
6. The unblinded result $E(m_3) = E(M)/E(r)$ is computed and output.

5 Robustness

The above protocol has three weaknesses with respect to robustness. First, it is possible for a cheater to publish ciphertext pairs for which it is not the case that the respective plaintexts add up to one; second, it is possible for a cheater to cause incorrect decryption; and third, it is possible for a cheater to publish a value which is not the sum of the plaintexts she received. We note that it is not possible to corrupt the computation in other places, as the blinding factor $E(r)$ applied in the first step is cancelled in the last, and both of these computations are performed "in public".

We address avoidance of the first and the third attack in the following two subsections, starting with how to guarantee correct one-splittings, followed by a method for guaranteeing correct addition of plaintexts. The second attack is easily avoided by use of proofs of correct exponentiation (see e.g., [8]).

5.1 Attaining Robustness I (Protocol $\mathcal{P}_2$)

Let us consider how to guarantee that a one-splitting is correctly performed. Let $(\mathcal{E}_{j1}, \mathcal{E}_{j2}, \mathcal{E}_{j3}, \mathcal{E}_{j4})$ be the previously described one-splittings. The players run the following protocol to verify once for each such one-splitting $1 \leq j \leq 2$:

3a. The two active parties compute a blinding factor $E(\rho_j)$.

3b. Let $\beta_{j\kappa} = \mathcal{E}_{j\kappa} E(\rho_j)$, for $1 \leq \kappa \leq 4$. Using directed decryption, the parties decrypt these ciphertexts, giving (as before) the first player the plaintext with $\kappa = 1$, the second that with $\kappa = 3$, and the third player the remaining two plaintexts.

3c. Using the blinded round-robin addition method, they compute the sum of the plaintexts they have been given. Here, we use the scheduling order $(1, 3, 2)$. This corresponds to a change in the order of the active players with respect to the main protocol. The resulting ciphertext is $E(B_j)$.

3d. The players determine if $E(B_j)$ and $E(\rho_j)$ correspond to the same plaintexts, using the *plaintext equality test* building block. They accept iff the plaintexts are equal.

As suggested by the enumeration of the above steps, this protocol is meant to be inserted in place of step 3 in protocol $\mathcal{P}_1$. We call the resulting protocol $\mathcal{P}_2$. Protocol $\mathcal{P}_2$ implements privacy, as stated in the following lemma, whose proof is sketched in Appendix A.

Lemma 1: Protocol $\mathcal{P}_2$ implements privacy. More precisely, we can construct a simulator Σ such that an adversary $\mathcal{A}$ controlling a minority of the players cannot distinguish the view of a real protocol run from the view generated by Σ, assuming that the adversary only corrupts a minority of participants, and that the DDH problem over $F^*[2^t]$ is hard. $\square$

If the parties accept in the above protocol, then the one-splitting must be correct with overwhelming probability; otherwise, somebody must have cheated. In other words, the protocol $\mathcal{P}_2$ for proving valid one-splitting has following the property, relating to the robustness of the final scheme.

Lemma 2: If the parties accept in the protocol described by steps 3a-3d, then with overwhelming probability, for each quadruple $(\mathcal{E}_{j1}, \mathcal{E}_{j2}, \mathcal{E}_{j3}, \mathcal{E}_{j4})$, $1 \leq j \leq 2$, the sum of the corresponding plaintexts is congruent to 1. $\square$

Note that the above protocol only *detects* cheating, but does not determine who cheated. In order to reveal the identity of a cheater, all the players publish their protocol-specific inputs, after which the computation of each player is verified by each other player, and the cheater pinpointed. This procedure is, of course, only performed in case that the above protocol results in a reject.

5.2 Attaining Robustness II (Protocol $\mathcal{P}_3$)

The protocol we called $\mathcal{P}_1$ is not robust, as it allows a cheating player to use an arbitrary value as input to the blinded round-robin addition step without being detected. Again, we can use the principles of repetition robustness to avoid this problem. More precisely, we can run the previously described protocols twice, using the same inputs but different random strings. The output can be shown (and will be shown) to be correct with an overwhelming probability if the two resulting ciphertexts correspond to the same plaintext values.

In fact, we do not have to run the partially robust protocol $\mathcal{P}_2$ twice. We can, instead, execute the protocol $\mathcal{P}_3$, which is as follows.

1. Run one instance of $\mathcal{P}_1$ and one instance of $\mathcal{P}_2$ on the same inputs, but using independent random strings.
2. Let $E(m_3)$ be the output of the above invocation of $\mathcal{P}_1$ and $E(m_4)$ be the output of the above invocation of $\mathcal{P}_2$. These are compared using the *plaintext equality test* building block. If the equality holds, then the players output $E(m_3)$. Otherwise they must perform a protocol $\mathcal{P}_4$ for identifying cheaters.

We note that the protocol $\mathcal{P}_4$, which we do not elaborate on, can use general multi-party computation, and therefore be computationally expensive. However, since it is only employed in what are presumably rare cases of cheating, this is not a concern. (In other words, we take a fast-track or "optimistic" approach to robustness.) It is easy to see that protocol $\mathcal{P}_3$ is correct and complete. Furthermore, as will be proven in Appendix A, it also implements privacy, and is robust. Thus we have the following two theorems.

Theorem 1: Assuming $\mathcal{P}_4$ is private (which will follow from its zero-knowledge properties), we have that protocol $\mathcal{P}_3$ also is private. More precisely, we can construct a simulator Σ such that an adversary $\mathcal{A}$ controlling exactly one of the players cannot distinguish the view of a real protocol run from the view generated by Σ. $\square$

Theorem 2: Protocol $\mathcal{P}_3$ is robust. That is, if $E(m_1)$ and $E(m_2)$ are the input ciphertexts, then the output of $\mathcal{P}_3$ will be $E(m)$, where $m \equiv m_1 + m_2$. This is under the assumption that the adversary only corrupts a minority of participants, and that the DDH problem over $F^*[2^t]$ is hard. $\square$

Acknowledgements

Many thanks to Daniel Bleichenbacher, Kevin McCurley and Moti Yung for helpful discussions.

References

1. D. Chaum, "Zero-Knowledge Undeniable Signatures," in *Advances in Cryptology – EUROCRYPT '90*, I Damgård, ed., pp. 458–464, Springer-Verlag, 1990. LNCS No. 473.
2. D. Chaum and H. van Antwerpen, "Undeniable Signatures," in *Advances in Cryptology – CRYPTO '89*, G. Brassard, ed., pp. 212–216, Springer-Verlag, 1989. LNCS No. 435.
3. D. Coppersmith, "Fast evaluation of logarithms in fields of characteristic two," IEEE Transactions of Information Theory, 30 (1984), 587–594.
4. T. ElGamal, "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," *IEEE Transactions on Information Theory*, vol. 31, pp. 469-472, 1985.
5. M. Franklin and M. Yung, "Communication Complexity of Secure Computation", in *Proc. 24th Annual Symp. on the Theory of Computation (STOC)*, pp. 699-710, ACM Press, 1992.
6. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems", in *Advances in Cryptology – EUROCRYPT '99*, J. Stern, ed., pp. 295–310, Springer-Verlag, 1999. LNCS No. 1592.
7. R.Gennaro, M.Rabin, and T.Rabin, "Simplified VSS and Fast-track Multiparty Computations with Applications to Threshold Cryptography", in *Proc. 1998 ACM Symposium on Principles of Distributed Computing (PODC)*, ACM Press, 1999.

8. M. Jakobsson and C.-P. Schnorr, “Efficient Oblivious Proofs of Correct Exponentiation,” in *Communications and Multimedia Security (CMS) ’99*, B. Preneel, ed., pp. 71–84, Kluwer Academic Publishers.
9. M. Jakobsson, “A Practical Mix,” in *Advances in Cryptology – EUROCRYPT ’98*, K. Nyberg, ed., pp. 448–461, Springer-Verlag, 1998. LNCS No. 1403.
10. M. Jakobsson, “Flash Mixing,” in *Proc. 1999 ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 83–89, ACM Press, 1999.
11. A. Herzberg, S. Jarecki, H. Krawczyk, M. Yung, “Proactive Secret Sharing or How to Cope With Perpetual Leakage,” in *Advances in Cryptology – Crypto ’95*, D. Coppersmith, ed., pp. 339–352, Springer-Verlag, 1995. LNCS No. 963.
12. K. McCurley, personal communication.
13. A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.
14. T.P. Pedersen, “Distributed Provers with Applications to Undeniable Signatures,” in *Advances in Cryptology – EUROCRYPT ’91*, D.W. Davies, ed., Springer-Verlag, pp. 221–242, 1991. LNCS No. 547.
15. T.P. Pedersen, “Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing,” in *Advances in Cryptology – CRYPTO ’91*, J. Feigenbaum, ed., pp. 129–140, Springer-Verlag, 1991. LNCS NO. 576.
16. C.-P. Schnorr, “Efficient Signature Generation for Smart Cards,” *Journal of Cryptology*, vol. 4, pp. 161-174, 1991.
17. A. Shamir, “How to share a secret”, *Communications of the ACM*, vol. 22, pp. 612-613, 1979.

A Proofs

Proof of Lemma 1: *(Sketch)*
Our approach is to show for each player that, given a view and a guess of an input pair $(\hat{m}_1, \hat{m}_2)$, these two are consistent with each other. If this is the case for each possible pair $(\hat{m}_1, \hat{m}_2)$, then each such pair is equally likely, given the view of the player, and thus, the protocol does not leak any information. For simplicity, we mark known, derived, and assumed quantities with a hat where it is not obvious from the context that they are known.

Player 1: The first active player knows M and can therefore compute $\hat{r} = \hat{M}/(\hat{m}_1 + \hat{m}_2)$. He also knows w_1^1, w_{21}^2, w_{22}^2. He has that

$$\begin{cases} \hat{\mu}_{11} = \hat{w}_1^1 w_{21}^1 \hat{r} \hat{m}_1 \\ \hat{\mu}_{21} = w_1^2 \hat{w}_{21}^2 \hat{r} \hat{m}_2 \\ \hat{\beta}_{14} = \hat{\overline{w}}_1^1 \overline{w}_{22}^1 \rho_1 \\ \hat{\beta}_{24} = \overline{w}_1^2 \hat{\overline{w}}_{22}^2 \rho_2 \end{cases}$$

For these four equations, there are five unknowns, w_{21}^1, w_{22}^1, w_1^2, ρ_1, and ρ_2. No matter what the first player’s view is, it is consistent with any pair $(\hat{m}_1, \hat{m}_2)$.

Player 2: Similarly, the second active player knows R_j. Thus, he knows $\hat{\rho}_j = \hat{R}_j$. He also knows w_{21}^1, w_{22}^1, w_1^2. He has that

$$\begin{cases} \hat{\mu}_{13} = \overline{w}_1^1 \hat{w}_{22}^1 r \hat{m}_1 \\ \hat{\mu}_{23} = \hat{\overline{w}}_1^2 w_{22}^2 r \hat{m}_2 \\ \hat{\beta}_{12} = w_1^1 \hat{\overline{w}}_{21}^1 \hat{\rho} \\ \hat{\beta}_{22} = \hat{w}_1^2 \overline{w}_{21}^2 \hat{\rho} \end{cases}$$

For these equations, there are four unknowns, w_1^1, w_{21}^2, w_{22}^2, and r. Again, we get that this view is consistent with $(\hat{m}_1, \hat{m}_2)$.

Player 3: Finally, the passive player knows the following eight equations:

$$\begin{cases} \hat{\mu}_{12} = w_1^1 \overline{w}_{21}^1 r \hat{m}_1 \\ \hat{\mu}_{14} = \overline{w}_1^1 \overline{w}_{22}^1 r \hat{m}_1 \\ \hat{\mu}_{22} = w_1^2 \overline{w}_{21}^2 r \hat{m}_2 \\ \hat{\mu}_{24} = \overline{w}_1^2 \overline{w}_{22}^2 r \hat{m}_2 \\ \hat{\beta}_{11} = w_1^1 w_{21}^1 \rho_1 \\ \hat{\beta}_{13} = \overline{w}_1^1 w_{22}^1 \rho_1 \\ \hat{\beta}_{21} = w_1^2 w_{21}^2 \rho_2 \\ \hat{\beta}_{23} = \overline{w}_1^2 w_{22}^2 \rho_2 \end{cases}$$

For the above eight equations, there are nine unknowns, corresponding to all of the values that went into making the two one-splittings (namely w_1^1, w_1^2, w_{21}^1, w_{22}^1, w_{21}^2, and w_{22}^2); and the blinding factors, r, ρ_1, and ρ_2. Therefore, the passive player's view is also consistent with any pair $(\hat{m}_1, \hat{m}_2)$, and thus, $\mathcal{P}_2$ is private. $\square$.

Remark 1. In this proof sketch and those that follow, we do not fully consider information that the players may derive from published ciphertexts. Under the DDH assumption, the semantic security of ElGamal assures that this information is negligible. We shall treat this issue formally in proofs provided in the full version of the paper.

Corollary 1: Each component of $\mathcal{P}_2$ is private, and in particular, $\mathcal{P}_1$ is.

Claim 1: Any composition of private protocols with independent random strings for the players is private.

Proof of Lemma 2: *(Sketch)*
Assume that there is a polynomial-time cheating algorithm $\mathcal{A}$ for generating the transcripts of steps 1-3 of $\mathcal{P}_2$, so that the plaintext sum is not congruent to 1, and the honest players accept in step 3d with a non-negligible probability. We will show how to use $\mathcal{A}$ to break the DDH assumption in $F^*[2^t]$. The input to the algorithm will be the ciphertexts that constitute the output from the honest players, namely those for the generation of $(\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3, \mathcal{E}_4)$, and those for the generation of $E(\rho)$. For $E(B)/E(\rho)$ to correspond to the plaintext 1 requires

that the plaintexts in step 3c add up to ρ. Assume that the portions held by the honest players are known by the adversary. If $\mathcal{A}$ could produce a share so that the sum of the shares equals ρ with a non-negligible probability, $\mathcal{A}$, together with simulations of the honest players (which takes a suspected value ρ as input), could be used to determine if $E(\rho)$ is an encryption of ρ with a non-negligible success probability. This would break the DDH assumption in $F^*[2^t]$, as it would show that the standard ElGamal encryption scheme is not semantically secure in this group. $\Box$.

Proof of Theorem 1: *(Sketch)*
Consider first the case in which no cheater is detected: Since $\mathcal{P}_1$ and $\mathcal{P}_2$ both are private, so must be a composition of the two. Consider now the case in which a cheater is detected. If a cheater is detected in step 3 of $\mathcal{P}_2$, then the entire protocol run is halted after each player reveals his random strings. Since no player has computed any function of his secret inputs at this point, that cannot leak any information, and this event must be simulable. The privacy of $\mathcal{P}_3$ therefore follows from the our choice of a good secure function evaluation protocol for $\mathcal{P}_4$. If the latter is zero-knowledge, $\mathcal{P}_3$ will also be zero-knowledge. $\Box$

Proof of Theorem 2: *(Sketch)*
We know from Lemma 2 that $\mathcal{P}_2$ is robust, and that if the players accept in this sub-protocol, that the one-splittings with an overwhelming probability are correct. The robustness of step 4 of $\mathcal{P}_1$ follows from the soundness of the proof protocol for proving correct decryption. We will now show that if the result of the addition of plaintexts (step 5 of $\mathcal{P}_1$) is corrupted, then it will be detected by the honest players with an overwhelming probability. In order for the results of the two computations (of $\mathcal{P}_1$ resp. $\mathcal{P}_2$) to be equal, we have the following: the adversary must add a value vr_1 in the addition step of the first protocol, and a value vr_2 in the second, where r_1 is the blinding factor used in the first protocol, and r_2 that used in the second. Following the argument in the proof of Lemma 2, this would allow him to break the DDH assumption on $F^*[2^t]$, as only encryptions of these values are available to him. Therefore, the output will, with overwhelming probability, only be accepted when it is correct. $\Box$