EFFICIENT OBLIVIOUS PROOFS OF CORRECT EXPONENTIATION

Markus Jakobsson
Information Sciences Research Center, Bell Labs, Murray Hill, New Jersey 07974
www.bell-labs.com/user/markusj

Claus Peter Schnorr
University of Frankfurt; work done while visiting Bell Labs.

Abstract We study the notion of *meta-proofs*, which, as the name indicates, are proofs about proofs. We employ the notion of meta-proofs to produce a highly efficient oblivous proof of correct exponentiation. It is minimum-knowledge independently of whether the input is valid or not, a property that does not hold for many other protocols (that are zero-knowledge only for valid inputs.) This has direct security implications to multi-party protocols, where the protocols we demonstrate – one interactive and one non-interactive – can be employed to obtain protocol robustness at a low cost. As a result of potential independent interest, we show how to turn any standard discrete log signature scheme into a scheme for proving equality of discrete logarithms. We demonstrate our method using the Schnorr signature scheme.

1. INTRODUCTION

In many applications, there is a need for parties to prove to each other that the correct computation was performed. Given today's choices of efficient primitives, this often boils down to proving that the intended exponentiation, in relation to some public key, was performed. In many multi-party protocols, whose robustness depend on these types of proofs, however, it is not known beforehand whether the relation holds or not. Therefore, if the proof for proving a correct exponentiation *requires* that the computation indeed was correctly performed, then such a protocol may leak important information when given *invalid* inputs. This in turn may endanger protocol properties, such as privacy, as it potentially allows attacks on the protocol.

B. Preneel (ed.), *Secure Information Networks*

This begs the question of how to design protocols that do not leak any information whether given valid or invalid inputs. Since the very aim of the protocol may be to determine whether the input is valid or not, that limits us to protocols consisting of two sub-protocols, one for valid inputs and the other for invalid inputs, such that the behavior of the distributed prover is *identical* for both sub-protocols. Such a protocol is called *oblivious*, since it does not require the protocol participants to know beforehand whether the input is of one type or another in order to correctly perform the computation.

Fujioka, Okamoto and Ohta [9] proposed the first protocol where the prover did not have to choose between two different sub-protocols. It is not clear, however, how to distribute their protocol, which is the main motivation in this paper. This was also the motivating element in the work by Jakobsson and Yung [10], who minted the term oblivious, and who proposed an oblivious multi-party protocol for determining whether a given exponentiation was correctly performed. Their protocol requires computation and communication logarithmic in the length of the security parameter, e.g., requires $O(k)$ rounds and exponentiations in order to reduce the failure probability to $O(2^{-k})$. Whereas this is not entirely precluding the use of their protocol, it does limit the number of applications.

By employing the notion of a *meta-proof*, we are able to limit the above mentioned costs to a low and constant cost. A meta-proof in our setting consists of two portions: (1) a "blinded" proof of the statement whose aim it is to prove or disprove – in our case "the exponentiation was correctly performed", and (2) a proof that the first proof was correctly performed. The first proof is blinded to avoid leaks of information; the second is employed to maintain soundness in the presence of the blinding. If both proofs succeed, the verifier concludes that the exponentiation was correctly performed. On the other hand, if the first proof fails and the second proof succeeds, this means that the exponentiation was *not* correctly performed.

We demonstrate an oblivious and computationally minimum-knowledge meta-proof for deciding valid exponentiation. We exhibit two versions, one that is interactive and based on standard protocols for verification of undeniable signatures [5, 6]; the other non-interactive. The non-interactive version can be based on any discrete log based signature scheme of a common format, e.g., [8, 12, 17]. We call such a proof a *DLEQ signature*, as it is both a signature and a proof of equality of discrete logs.

The DLEQ signature is a result of potential independent interest. In order to obtain this result, we exhibit a first transformation method that

takes a quadruple (g, y, m, s) as input, and generates the pair (G, Y) such that G is a generator and Y is a public key. We also exhibit a related second transformation method that generates a secret key X, for which $Y = G^X$. Based on the random oracle assumption, we show that it is only possible to determine the secret key X if $log_g y = log_m s$.

Using these new parameters, the prover can use a standard discrete log based signature scheme to convince the verifier that the relationship between the discrete logarithms holds. This is done simply by the prover generating a signature on some message using G as a generator, Y as a public key, and X as the corresponding secret key. This signature is given to the verifier. If the signature is valid, the verifier will conclude that $log_g y = log_m s$, since the prover with overwhelming probability must have known X. We demonstrate our new method using Schnorr signatures [17] and a variant thereof.

The method for transforming the input elements to a generator and to public and secret keys draws on work by Bellare, Garay and Rabin [1]. They introduced a construction that raises different factors of a product to different powers in order to improve the efficiency of batch verification of exponentiation. We use the same trick, but for a different purpose, namely to "lock together" different input components. We prove that it is only feasible to determine the secret key corresponding to these new aggregate components if the input components have a given discrete log relationship.

Our contribution is threefold: First, we introduce the notion of meta-proofs to improve the efficiency of protocols. Second, we use this new notion to develop an efficient method for verifying the correctness of exponentiation, with numerous applications within multi-party protocol design. Third, we show how to transform inputs that consists of pairs with the same discrete logarithm relationships into an output that corresponds to the public information needed in signature schemes. It is possible to generate signatures using these new parameters if and only if the common discrete log of the input pairs is known. The resulting non-interactive proof is useful in its own right to provide robustness of multi-party protocols.

Outline. We start in section 2 by reviewing related work. We continue in section 3 by specifying the problem we strive to solve. Then, in section 4, we present an oblivious protocol to decide whether an input corresponds to a correct exponentiation. This is followed in section 5 by the introduction of our new key transformation method, which in conjunction with standard signature schemes is used to make the above

oblivious decision protocol non-interactive. In section 6 we state the properties of our schemes; these are proven in the Appendix.

2. RELATED WORK

The effort of determining whether a given quadruple (g, y, m, s) is such that $log_g y = log_m s$ was started by Chaum and Antwerpen [5], who studied the problem in the context of verifying the validity of undeniable signatures (from which we borrow the above denotation). In [5] a method for proving validity of undeniable signatures was proposed. This was improved in [6], resulting in a method that was zero-knowledge *for valid inputs.*

It was assumed in this setting that the prover would know whether the signature is valid or not. This is crucial, as by running the above mentioned zero-knowledge proof for an *invalid* input, the prover in fact *leaks* what the corresponding *valid* signature is. For invalid inputs, a standard "distinguishing protocol" (similar to what has been proposed to prove graph non-isomorphism) had to be used.

In [13], Pedersen showed how to distribute the protocol for proving validity of undeniable signatures (or, in our terminology, of correct exponentiation), but still under the assumption that the prover already knew whether the input corresponds to a valid undeniable signature or not. However, this is often an unrealistic assumption, since it is circular in that it requires the prover to run the protocol for deciding the validity of the input *before* the proper protocol could be selected.

In [9], Fujioka, Okamoto and Ohta introduced a protocol that was symmetric in the sense that it contained two for the prover identical portions, one for proving validity of undeniable signatures, the other for proving invalidity. It is not clear, however, how to distribute their protocol. The work on this type of symmetric two-component protocols was continued by Jakobsson and Yung [10], who demonstrated an alternative proof protocol – which allows the distribution of the prover – and extended this to a setting in which the prover cannot *learn* whether the input is in the language or not. (Here, we are only concerned with the prover not having to *know* this fact beforehand, and allow the prover to learn this bit of information.)

Another thread of work of interest to our result is that on random oracles, as we use results for random oracles in our transformation scheme. We refer to [2, 3, 14, 15, 16] for a careful treatment of issues relating to random oracles and their significance to signature schemes.

3. PROBLEM SPECIFICATION

A quadruple (g, y, m, s) is given to a set of participants that share the secret key x corresponding to the public key $y = g^x$. Here – and onwards – all computation is assumed to be modulo p, unless otherwise stated, where p is a large prime such that $p = lq+1$ for an integer l and another large prime q.

It is the goal of these participants to determine whether or not $s = m^x$ holds. For simplicity of the protocol description it is also assumed that x is shared using a (k, n) threshold scheme, as described by Shamir [18].

Assumptions. The computational assumption we make is that a random quadruple (g, g^x, m, m^x) cannot be distinguished from (g, g^x, m, R) for a random $R = m^r$, unless x is known. This assumption is known as the *Decision Diffie-Hellman* assumption.

Requirements. We say that a quadruple (g, y, m, s) is *in the language of valid quadruples* iff $log_g y = log_m s$. This is with respect to a given pair of prime moduli (p, q) of the assumed format.

We present a protocol for deciding language membership of given quadruples (g, y, m, s). We require our solution to be *correct* (all the computation can be performed by the participants involved), *sound* (the decision made corresponds to the true language membership with an overwhelming probability) and *minimum-knowledge* (the protocol leaks no information except for the desired one bit result.) To expand a little on the latter requirement, our protocols will be *computational* minimum-knowledge, which means that given an oracle for language membership, it is possible in p-time to simulate transcripts that cannot be distinguished with a non-negligible probability by any p-time participant from real protocol transcripts for the corresponding proof. Moreover, the protocol is *oblivious*, i.e., the prover executes the same protocol for input quadruples in the language as for those that are not.

As a protocol component of independent interest, we develop a transformation method that takes as input a quadruple (g, y, m, s), such that $log_g y = log_m s$, and outputs a pair (G, Y) such that G is a new generator, and Y is a new public key. A second transformation protocol outputs $X = log_G Y$ given the quintuple (g, y, m, s, x), where $x = log_g y = log_m s$. The new parameters can be used in any correct and sound discrete log based signature scheme of a common type. We require that the public key and secret key transformation protocols are *correct*, i.e., if the generated keys are used in a signature scheme in the manner shown, then the signature proof succeeds if the input parameters to the transformation

protocol have the same pairwise discrete log relation. We also demand that the transformation protocols are *sound*, i.e., the verifier of the signature protocol in which they are used will reject the signature with an overwhelming probability if the input parameters to the transformation protocol do *not* have the pairwise discrete log relation.

4. AN OBLIVIOUS DECISION PROOF

Let us consider the non-distributed version of the proof for simplicity of denotation. Of course, this is a setting where oblivious protocols are not needed for security reasons, since the prover can decide whether to use the protocol for language membership or non-membership before the start of the protocol. The protocol we present is easily distributed.

The prover is given a quadruple (g, y, m, s), and needs to determine – and prove – whether $log_g y = log_m s$. The prover knows x, the discrete logarithm of y w.r.t. g.

The protocol is as follows:

1. **Setup.** The prover selects a number $a \in_u Z_q$ uniformly at random.

2. **First-order proof.** The prover generates and outputs what corresponds to a first order proof, i.e., the triple $(\overline{s}, \overline{\sigma}, \overline{m}) = (s^a, m^{ax}, m^a)$. A verifier of this first proof accepts iff $\overline{s} = \overline{\sigma}$.

3. **Second-order proof.** The prover proves that $log_m \overline{m} = log_s \overline{s}$ and that $log_g y = log_{\overline{m}} \overline{\sigma}$. The verifier of this second proof accepts iff both equations are found to hold.

4. **Decision.** The verifier outputs `exponentiation valid` if he accepted both the first and second order proofs.
   He outputs `exponentiation invalid` if he rejected the first order proof and accepted the second order proof.
   Otherwise, he outputs `cheating prover`.

Using a proof protocol for undeniable signatures [5, 6] to prove equality of discrete logs, an interactive version of the above protocol is obtained. In the next section, we consider how to construct a simple and efficient non-interactive protocol for the same, using any common discrete-log based protocol.

5. KEY TRANSFORMATION & DLEQ SIG

In this section we detail a non-interactive proof protocol for performing the two proofs of equality of discrete logs needed in the protocol in the previous section. This is obtained in two steps. First, we introduce our key transformation protocol, which takes an input with a certain claimed discrete log relation and successfully produces an output consisiting of public and secret keys of a certain format if and only if the claimed relation holds. Second, we show how these new parameters can be used in standard signature schemes of a common format (we demonstrate the method for Schnorr signatures). Here, the DLEQ proof is said to succeed if and only if the corresponding signature is valid.

Again, we only consider the non-distributed version of the protocols, in order to simplify the notation, and given that the change needed to obtain the distributed version is trivial.

In the following, we consider two different scenarii; one in which it is impossible that $log_g m$ is known to the prover (e.g., m is chosen as a hash of a message and the system parameter g, or chosen by the verifier); the second in which the prover may (but does not need to) know the value $log_g m$.

5.1 RELATION UNKNOWN

We consider the solution that can be employed to efficiently prove equality of discrete logarithms of the type $log_g y = log_m s$ when it is impossible that the prover knows $log_g m$:

Public Key Transformation Scheme. The tranformation algorithm takes as input the quadruple (g, y, m, s). Two randomizing coefficients are computed. These are $e_j = hash(g, y, m, s, j)$, for $j \in \{1, 2\}$, where *hash* is an arbitrary hash function that can be modelled by a random oracle. It then pairwise "locks together" the components of the input in the following manner: $G = g^{e_1} m^{e_2}$, and $Y = y^{e_1} s^{e_2}$. The transformation algorithm outputs the pair (G, Y), where G is denoted the *new generator* and Y is denoted the *new public key.*

Secret Key Transformation Scheme. There is a similar transformation scheme between secret keys. This is much more straightforward, however, as it simply involves setting the output secret key to the input secret key, or $X = x$. We denote X the *new secret key.*

We note that the above can easily be extended to any polynomial number of components, without affecting the size of the resulting output values. Once we have generated the above values, these can be used in

a standard signature scheme. Let us review how Schnorr signatures [17] are generated:

Standard Schnorr signatures. The prover selects a value $k \in Z_q$ uniformly at random. He computes $r = g^k$, and the value $t = k - cx \bmod q$, where $c = hash(\mu, r)$, for a message μ to be signed, and x is the secret key of the signer, with a corresponding public key $y = g^x$. The prover outputs (r, t) as a signature on m. The signature is verified by checking that $r = y^c g^t$ for $c = hash(\mu, r)$.

The Schnorr signature scheme can be directly used to prove the equality of discrete logarithms by using (G, Y, X) instead of (g, y, x). The message μ is irrelevant in this setting. Similarly, any other signature scheme with this general structure may be employed.

5.2 RELATION POTENTIALLY KNOWN

In the following, we assume the scenario in which it is possible that the prover knows $log_g m$, for a relation $log_g y = log_m s$ that he wants to prove. The reason that the above protocol cannot be employed is that the prover can generate a valid signature for any (g, y, m, s) where he knows $(log_g y, log_g m, log_g s)$, whether or not $log_g y = log_m s$, as he will always be able to generate the secret key required.

The solution is closely related to the previously shown solution, but for some small differences:

Key Transformation Scheme. Instead of producing only one pair (G, Y) as above, the prover generates *two* such pairs. In the simple case, where we only want to prove equality of two discrete logs, we can set $(G, Y, M, S) = (g, y, m, s)$ and $X = x$. If more relations are to be shown to hold, we can let one of the pairs, e.g., (G, Y), correspond to all but one of the components of the proof. As above, we would then seperate individual components by raising them to random exponents. The second pair, (M, S) is set to the remaining two values, e.g., $(M, S) = (m, s)$.

Using a method employed in [11], a pair of related signatures can be constructed to show equality of two discrete logarithms, e.g., $log_G Y = log_M S$. We show how this can be done using the Schnorr signature scheme as a basis:

Siamese Schnorr Signature. The prover takes two generators, (g_1, g_2) as input, and one secret key x. He selects a value $k \in Z_q$ uniformly at random and computes $r_1 = {g_1}^k$, $r_2 = {g_2}^k$, and $t = k - cx \bmod q$, where $c = hash(\mu, r_1, r_2)$, for a message μ to be signed, and a secret key x of the signer. The prover outputs (r_1, r_2, t) as a signature on m. (Alternatively, corresponding to the methods for *short* Schnorr signatures, he may output (c, t), slightly altering the verification method.)

The signature is verified by checking that $r_1 = {y_1}^c {g_1}^t$ and $r_2 = {y_2}^c {g_2}^t$ for $c = hash(\mu, r_1, r_2)$.

6. PROPERTIES

Our new schemes have the following properties: The oblivious decision proof for correct exponentiation is *computational minimum-knowledge* (Theorem 1,) *correct* (Theorem 2,) and *sound* (Theorem 3.) Herein, we do not specify whether the interactive or non-interactive sub-protocol for proving valid exponentiation will be used. We then prove that our transformation protocols – which in conjuction with standard signature schemes can be used to produce proofs of equality of discrete logarithms, and in extension, of correct exponentiation – are *correct* (Theorem 4,) and *sound* (Theorem 5.) These properties are proven in the Appendix.

7. CONCLUSION

We have proposed an efficient method for performing oblivious proofs of correct exponentiation. Such proofs are well suited to be used by a distributed prover to determine, and prove, whether a relation holds. This is so since the protocol for proving language membership is identical to that for proving non-membership, and so, it is not necessary for the servers to know this fact before the start of the protocol. This property avoids potential leaks of information. Our methods employs a meta-proof structure, i.e., uses two sub-proofs to prove or disprove the statement. In the first proof, the prover attempts to prove correctness of the statement, corresponding to language membership, and in the second, he proves that the first proof was performed correctly. A verifier concludes that the statement in question is correct if both the proofs succeed, and that the statement is incorrect if only the second proof succeeds. As a partial result we show how to employ signature schemes of a common type for proving equality of discrete logarithms.

References

[1] M. Bellare, J. Garay, T. Rabin, "Fast Batch Verification for modular Exponentiation and digital Signatures," Eurocrypt 98, pp. 236–250.

[2] M. Bellare and P. Rogaway, "Random Oracles are Practical: a Paradigms for Designing Efficient Protocols," Proc. of the 1st ACM Conference on Computer Communication Security, pp. 62–73, 1993.

[3] R. Canetti, O. Goldreich and S. Halevi, "The Random Oracle Methodology, Revisited," Proc. STOC'98, ACM Press, pp. 209–218, 1998.

[4] D. Chaum, "Blind Signatures for Untraceable Payments," Advances in Cryptology - Proceedings of Crypto '82, pp. 199–203.

[5] D. Chaum, H. Van Antwerpen, "Undeniable Signatures," Advances in Cryptology - Proceedings of Crypto '89, pp. 212–216.

[6] D. Chaum, "Zero-Knowledge Undeniable Signatures," Eurocrypt '90, pp. 458–464.

[7] A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung, "How to Share a Function Securely," STOC '94, pp. 522–533.

[8] T. ElGamal "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," Crypto '84, pp. 10–18.

[9] A. Fujioka, T. Okamoto, K. Ohta, "Interactive Bi-Proof Systems and Undeniable Signature Schemes," Eurocrypt '91, pp. 243–256.

[10] M. Jakobsson, M. Yung, "Proving Without Knowing: On Oblivious, Agnostic and Blindfolded Provers," Crypto '96, pp. 186–200.

[11] M. Jakobsson, K. Sako, R. Impagliazzo, "Designated Verifier Proofs and Their Applications," Eurocrypt '96, pp. 143–154.

[12] National Institute for Standards and Technology, "Digital Signature Standard (DSS)," Federal Register Vol 56(169), Aug 30, 1991.

[13] T.P. Pedersen, "Distributed Provers with Applications to Undeniable Signatures," Advances in Cryptology - Proceedings of Eurocrypt '91, pp. 221–242.

[14] D. Pointcheval and J. Stern, "Security Proofs for Signature Schemes," Proc. Eurocrypt'96, LNCS 1070, Springer-Verlag, pp. 387–398, 1996.

[15] D. Pointcheval and J. Stern, "Provably Secure Blind Signature Schemes," Proc. Asiacrypt'96, LNCS 1163, Springer Verlag, pp. 387-393, 1996.

[16] D. Pointcheval, "Strengthened Security for Blind Signatures," Proc. Eurocrypt'98 LNCS 1403, Springer Verlag, pp. 391 - 405, 1998.

[17] C.P. Schnorr, "Efficient Signature Generation for Smart Cards," Advances of Cryptology, Proceedings of Crypto '98, pp.239–252.

[18] A. Shamir, "How to Share a Secret," Communications of the ACM, Vol. 22, 1979, pp. 612–613.

Appendix: Proofs

Theorem 1: If the Decision Diffie Hellman assumption holds, then our oblivious protocol for deciding correct exponentiation is computational *minimum-knowledge*, i.e., given a bit corresponding to the desired output of the verifier (`exponentiation valid` or `exponentiation invalid`, there is a simulator whose transcripts cannot be distinguished by a p-time verifier from those generated by a real prover.

Lemma 1: Consider a correct computational zero-knowledge proof $(\mathcal{P}, \mathcal{V})$ of language membership for quadruples (g, y, m, s). Here, $\mathcal{P}$ is the prover, $\mathcal{V}$ a p-time verifier, and $\mathcal{S}$ a p-time simulator of $\mathcal{P}$. Assume that $\mathcal{V}$ does not know the discrete logarithms $log_g y$ vs. $log_m s$. Interacting with $\mathcal{S}$, it is infeasible for $\mathcal{V}$ to determine whether (g, y, m, s) is in the language or not, or $\mathcal{V}$ can be used as a black box to break the Decision Diffie-Hellman assumption.

Proof of Lemma 1:
We prove this by showing that for a valid quadruple (i.e., one in the language), there exist a simulator of valid transcripts; and that it is impossible to distinguish transcripts of valid quadruples from transcripts of invalid quadruples. Let (g, y, m, s) be a quadruple for which we want to determine language membership. $\mathcal{S}$ is an simulator of $\mathcal{P}$, which is a prover for proving – not deciding – language membership of the above type. We do not specify how $\mathcal{P}$ works here, but know that it exists, given that the protocol specified by $\mathcal{P}$ is zero-knowledge. We give the quadruple to $\mathcal{S}$, who interacts with $\mathcal{V}$ to prove that (g, y, m, s) is in the language – whether this is true or not. Given that the proof is assumed to be correct, $\mathcal{V}$ will only reject a proof with a non-negligible probability. Since the proof is computational minimum-knowledge *when*

given a query in the language, and $\mathcal{S}$ is a simulator for the proof, we know that it is infeasible for a p-time limited verifier $\mathcal{V}$ to distinguish a simulation from a real proof. Therefore, $\mathcal{V}$ will only reject a simulated proof of a *valid* quadruple with a negligible probability. Assume now that $\mathcal{V}$ will reject a simulation in which (g, y, m, s) is not in the language with a non-negligible probability. Then, we can use $\mathcal{V}$ to determine language membership of quadruples (g, y, m, s) of the valid format, by letting $\mathcal{V}$ interact with $\mathcal{S}$ on the given input. This would break the Decision Diffie-Hellman assumption. Therefore, we can conclude that it is not possible for $\mathcal{V}$ to determine language membership *when interacting with a simulator.*

Proof of Theorem 1:
Consider the following simulator $\mathcal{SIM}$: Given an input $(g, y, m, s, result)$, where *result* is the desired output of the verifier, the simulator performs the following:

- *result* = `exponentiation valid`
  Select $\alpha \in_u Z_q$, set $\overline{m} = m^\alpha$ and $\overline{\sigma} = \overline{s} = s^\alpha$.
- *result* = `exponentiation invalid`
  Select $\alpha, \beta \in_u Z_q$, set $\overline{m} = m^\alpha$, $\overline{s} = s^\alpha$, and $\overline{\sigma} = s^\beta$.

$\mathcal{SIM}$ sends $(\overline{\sigma}, \overline{s}, \overline{m})$ to the verifier. Then, it calls the simulator $\mathcal{S}$ of lemma 1 for proving validity of the quadruple $(g, y, \overline{m}, \overline{\sigma})$, letting $\mathcal{S}$ interact with the prover. Finally, $\mathcal{SIM}$ proves to the verifier that $(s, \overline{s}, m, \overline{m})$ is in the language of valid quadruples. (Since $\mathcal{SIM}$ knows α, it needs not simulate this proof.)
The verifier compares $\overline{s}$ and $\overline{\sigma}$, and accepts the first-order proof if $\overline{s} = \overline{\sigma}$, which happens with an overwhelming probability if and only if we have that the input to the simulator was *result* = `exponentiation valid`. Next, the verifier verifies the second-order proofs. According to Lemma 1, he will accept the simulated proof with an overwhelming probability. Obviously, he will accept the real protocol with an overwhelming probability, since the sub-protocol for proving valid exponentiation is assumed to be correct. Therefore, the verifier will accept the second-order proof with an overwhelming probability. Consequently, the verifier will output the decision *result* and halt.

Theorem 2: The oblivious protocol is *correct*, i.e., all computation can be performed by the participants, and the expected output is produced if all the participants are honest.

Proof of Theorem 2:
We only consider the correctness of the main protocol, and refer to the proof of theorem 4 for the correctness of the DLEQ signature. We begin by establishing that all the computation can be performed by the participants involved:
First, it is clear that the prover can perform the set-up, as this only involves picking a number at random. Also, the prover can perform the first-order proof, since the prover (potentially distributively) knows both a and x, and therefore will be able to produce the triple $(\overline{s}, \overline{\sigma}, \overline{m})$. Obviously, the verifier will be able to perform the computation of the first-order proof; this is a simple equality check.
Second, it is clear that both the prover and the verifier can perform the computation for a general formulation of the second-order proof, as the prover knows the secrets involved (a resp. x), and no secret information is required by the verifier. Finally, it is clear that the verifier can perform the last step, which only entails making a decision given the decisions from the two sub-proofs.
Now, considering the outputs in a situation where all the participants are honest, we see that the verifier will accept the first-order proof only when $\overline{s} = \overline{\sigma}$, which for an honest prover will occur exactly when $s = m^x$, i.e., when the input quadruple is in the language of valid exponentiations.
Given the general version of the second-order proof, which will always cause the verifier to accept in the setting where all participants are honest, we see that the final desicion output by the verifier will be correct.

Theorem 3: The protocol is *sound*, i.e., it is infeasible for a dishonest prover to make the verifier output `exponentiation valid` for an input quadruple not in the language, or `exponentiation invalid` for an input quadruple in the language.

Proof of Theorem 3:
Again, we consider only the general version of the oblivious proof. Given that the second order proof is sound (we refer to [6] resp. the proof of theorem 5 for this) we have that the prover cannot cheat in the first level proof. This holds since the second order proof governs the correctness of the first-level proof. Thus, the output corresponds to the result of the first-order proof, and we see that the verifier with overwhelming probability will accept if and only if the input is in the language.

Theorem 4: The DLEQ signature is *correct*, i.e., when used in conjunction with a sound and correct signature scheme, the resulting signature will be valid with an overwhelming probability if the input (g, y, m, s) to the transformation protocol is such that $log_g y = log_m s$.

Proof of Theorem 4:
For concreteness, we consider the particular implementation of the DLEQ signature that uses the Schnorr signature scheme. The general proof for an arbitrary signature scheme of the proper type follows easily. For the input quadruple (g, y, m, s), we have that $Y^c G^t = y^{e_1 c} s^{e_2 c} g^{e_1 t} m^{e_2 t} = g^{e_1(t+cx)} m^{e_2(t+cx)} = G^k = r$. Therefore, since all the values involved can be computed by the all the participants, we have that the protocol is correct.

Theorem 5: The transformation protocol is *sound*, i.e., when used in conjunction with a sound and correct signature scheme, the resulting signature will be valid only if the input (g, y, m, s) to the transformation protocol is such that $log_g y = log_m s$.

Proof of Theorem 5:
We prove this theorem by giving a reduction to computing discrete logarithms. Assume that we want to determine $log_g m$. Call this unknown value z. We know that e_1 and e_2 are random and uniformly distributed, following the random oracle assumption. We also see that $G = g^{e_1} m^{e_2} = g^{e_1 + e_2 z}$, and that $Y = y^{e_1} s^{e_2} = g^{x_1 e_1} m^{x_2 e_2} = g^{e_1 x_1 + e_2 x_2 z}$, for $y = g^{x_1}$ and $s = m^{x_2}$. We assume that $x_1 \neq x_2$, but that there exists a p-time algorithm $\mathcal{A}$ that can compute X, given the public input (g, y, m, s). This value $X = log_G Y = \frac{e_1 x_1 + e_2 x_2 z}{e_1 + e_2 z}$. We now treat this algorithm $\mathcal{A}$, which must succeed with a non-negligible probability, as a blackbox. We choose values x_1 and x_2 randomly, and compute s and y from m and g, which constitute the input to our algorithm. Then, knowing x_1, x_2, e_1, e_2 and the output X of the blackbox, we can compute z, which is the discrete log of m with respect to g. Therefore, if the discrete log problem is hard and the random oracle assumption holds, then we reach a contradiction. Thus, we conclude that the transformation protocol is sound.