Non-Interactive Zero-Knowledge from Homomorphic Encryption

Ivan Damgård[1], Nelly Fazio[2⋆], and Antonio Nicolosi[2⋆]

[1] Aarhus University, Denmark[⋆⋆],
ivan@brics.dk
[2] Courant Institute of Mathematical Sciences, New York University, NY, USA
{fazio,nicolosi}@cs.nyu.edu

Abstract. We propose a method for compiling a class of Σ-protocols (3-move public-coin protocols) into non-interactive zero-knowledge arguments. The method is based on homomorphic encryption and does not use random oracles. It only requires that a private/public key pair is set up for the verifier. The method applies to all known discrete-log based Σ-protocols. As applications, we obtain non-interactive threshold RSA without random oracles, and non-interactive zero-knowledge for NP more efficiently than by previous methods.

1 Introduction

In a zero-knowledge proof system, a prover convinces a verifier via an interactive protocol that some statement is true *i.e.,* a given word x is in some given language L. The verifier must learn nothing beyond the fact that the assertion is valid. Zero-knowledge is an extremely useful notion and has found innumerable applications.

One efficient variant is known as Σ-protocols, which are three-move protocols where conversations are tuples of the form (a, e, z) and e is a random challenge sent by the verifier. A large number of such protocols are known for languages based on discrete logarithm problems, such as Schnorr's protocol [16] and many of its variants, *e.g.*, for proving that two discrete logs are equal [4]. This last variant is useful, for instance, in threshold RSA protocols [17], where a set of servers hold shares of a private RSA key, and clients can request them to apply the private key to a given input. The Σ-protocol is used here by the servers to prove that they follow the protocol.

One well-known technique for making Σ-protocols non-interactive is the Fiat-Shamir heuristic [11], where e is computed by the prover himself as a hash of the statement proved and the first message a. In the random oracle model, where the hash function is replaced by a random function, this can be shown to work.

⋆ Research conducted while visiting BRICS.

⋆⋆ Supported by BRICS, Basic Research in Computer Science, Center of the Danish National Research Foundation, and FICS, Foundations in Cryptography and Security, funded by the Danish Research Council.

However, it is not in general possible to instantiate the random oracle with a concrete function and have the security properties preserved (*cf.* [12]). In other words, a proof in the random oracle model does not guarantee security in the real world.

Cramer and Damgård [7] suggest a different type of proof for equality of discrete logarithms in the *secret-key zero-knowledge* model, where prover and verifier are assumed to be given private, but correlated secret keys initially. These proofs can be applied to build non-interactive threshold RSA protocols without random oracles, but unfortunately, it is required that *every* client using the system must have keys for the proofs set up with *every* server. This seems quite impractical in many cases, due to the large amount of interaction and secure memory needed to set up and manage these keys. Moreover, [7] does not include any protocols for more general statements (such as NP-hard problems).

In this paper, we present a technique to compile a class of Σ-protocols into efficient non-interactive protocols, in the registered public-key model [1]. This model includes a trusted functionality for setting up a private/public key pair individually for each player (in fact, we only need this for the verifiers). Hence, unlike [7], the key setup is not tied to a particular prover/verifier pair: it can be implemented, for instance, by having the verifier send her public key to a trusted "certification authority" who will sign the key, once the verifier proves knowledge of her private key. Now, any prover who trusts the authority to only certify a key after ensuring that the verifier knows her private key, can safely (*i.e.,* in zero-knowledge) give non-interactive proofs to the verifier.

Our technique requires homomorphic public-key encryption such as Paillier's cryptosystem [15], and it preserves the communication complexity of the original protocol up to a constant factor. This is in contrast to the NIZK construction of Barak et al. [1] for the registered public-key model, which provides a much less efficient transformation from CCA-encryption and ZAP's [10].

The zero-knowledge property of our protocols is unconditional, whereas the soundness is based on an assumption akin in spirit to "complexity leveraging" [3]. More precisely, we assume that, by choosing large enough keys for the cryptosystem, the problem of breaking it can be made much harder than the problem underlying the Σ-protocol (for a particular meaning of "much harder" that we formalize in the paper).

An immediate consequence of our results is non-interactive threshold RSA and discrete-log based cryptosystems without random oracles, and assuming only that each client has a registered key pair. In the context of threshold cryptography where keys must be set up initially anyway, this does not seem like a demanding assumption. Our protocols are as efficient as the best known previous solutions (that required random oracles) up to a constant factor.

Another consequence is efficient non-interactive zero-knowledge arguments for circuit satisfiability, and hence for NP (in the registered public-key model). Namely, the prover commits to his satisfying assignment using a bit-commitment scheme for which appropriate efficient Σ-protocols exist. Then, using well-known techniques, for instance from [6], he could prove via a Σ-protocol that the committed bits satisfy the circuit. Compiling this protocol using our technique leads to the desired non-interactive protocol, whose communication complexity is essentially $O(ks_c)$ bits, where s_c is the size of the circuit, and k is the security parameter. This compares favorably to the solution of Kilian and Petrank [14] in the common random string model, which have complexity $O(k^2 s_c)$, even when using similar algebraic assumptions as we do here. Recently, Groth et al. [13] proposed non-interactive zero-knowledge proofs for NP in the common reference string model based on a specific assumption on bilinear groups, and with the same communication complexity as our protocol. This result is incomparable to ours: [13] uses a more conventional setup assumption and does not need a complexity-leveraging type of cryptographic assumption. On the other hand, it needs to assume that the statement shown by the prover is chosen independently from the reference string; in our model, the prover may see the verifier's public key first and then attempt to prove any theorem of his choice.

2 Preliminaries

We start by introducing some concepts and assumptions that will be useful later.

2.1 Problem Generators and a Complexity Assumption

A *problem generator* $\mathcal{G}$ is a pair $\mathcal{G} = \langle G, g \rangle$, where G is a probabilistic polynomial-time algorithm and $g : \{0,1\}^* \rightarrow \{0,1\}^*$ is an arbitrary (and possibly non-efficiently computable) function. On input 1^k, algorithm G outputs a string u, which we call an *instance*; we refer to $g(u)$ as the *solution* to u, and we require that $g(u)$ has length polynomial in k. For instance, u might be the concatenation of a public key and a ciphertext while $g(u)$ is the corresponding plaintext. We will only be considering problems with unique solutions, since that is all we need in this paper.

We will say that a probabilistic algorithm A *breaks* $\mathcal{G} = \langle G, g \rangle$ *on instances of size* k, if setting $u \stackrel{r}{\leftarrow} G(1^k)$ and $y \stackrel{r}{\leftarrow} A(1^k, u)$, results in $y = g(u)$ with non-negligible probability. We will be looking at the running time of A as a function only of its first argument 1^k; notice that A will not always be restricted to time polynomial in k.

We define that a probabilistic algorithm A *completely breaks* $\mathcal{G} = \langle G, g \rangle$ *on instances of size* k by considering the same experiment: Set $u \stackrel{r}{\leftarrow} G(1^k)$ and $y \stackrel{r}{\leftarrow} A(1^k, u)$; however, this time we demand that there exists a polynomial P such that, except with negligible probability (over the random choices of G), and for all large enough k, we have $Pr(y = g(u)|\ u) \geq 1/P(k)$, where the last probability is only over the random choices of A. In other words, A should be able to solve (almost) any instance u with good probability.

Definition 1. *Consider two problem generators $\mathcal{G}$ and $\mathcal{H}$ and let f be a polynomial. We say that $\mathcal{H}$ is f-*harder *than $\mathcal{G}$ if there exists a probabilistic algorithm A running in time $T(k)$ such that A completely breaks $\mathcal{G}$ on instances of size k,* *but no algorithm running in time $O(T(k) + poly(k))$ breaks $\mathcal{H}$ on instances of size $f(k)k$ or larger.*

In other words, completely breaking $\mathcal{G}$ on instances of size k requires time $T(k)$, but given a similar amount of time, there is no significant chance to break $\mathcal{H}$—where, however, the $\mathcal{H}$-instances to be solved have size at least $f(k)k$. Note that if $T(k)$ is polynomial in k, then $O(T(k) + poly(k)) = poly(k)$ and the definition amounts to say that $\mathcal{H}$ generates instances that are hard in the usual sense. But if $T(k)$ is superpolynomial, more is required about the hardness of $\mathcal{H}$-instances—essentially that the complexity of breaking $\mathcal{H}$ grows "fast enough" with the security parameter k.

For problem generators $\mathcal{F}, \mathcal{G}$, we will say that $\mathcal{F}$ is *as easy as* $\mathcal{G}$, if there exists an algorithm that completely breaks $\mathcal{F}$ on instances of size k in time polynomial in k, plus a constant number of oracle calls to any algorithm that completely breaks $\mathcal{G}$. The lemma below now follows trivially from the above definitions:

Lemma 1. *Let $\mathcal{F}, \mathcal{G}, \mathcal{H}$ be problem generators. If $\mathcal{F}$ is as easy as $\mathcal{G}$ and $\mathcal{H}$ is f-harder than $\mathcal{G}$, then $\mathcal{H}$ is also f-harder than $\mathcal{F}$.*

As an example, consider the following problem generator $\mathcal{G}_{dlog} = \langle G_{dlog}, g_{dlog} \rangle$: on input 1^k, G_{dlog} outputs an instance $u \doteq (p, p', g, h)$, where p, p' are primes, p' is k-bit long, $p = 2p' + 1$, g is an element of Z_p^* of order p' and $h = g^w \bmod p$, for some $w \in Z_{p'}$. In this case, the solution is $g_{dlog}(u) \doteq w$.

As another example, let $\mathcal{H}_{Paillier} = \langle H_{Paillier}, h_{Paillier} \rangle$, where $H_{Paillier}(1^k)$ outputs a k-bit RSA modulus n along with $c \doteq (1 + n)^w r^n \bmod n^2$ (*i.e.*, c is a Paillier encryption of w), where w is chosen in some given interval. Here, $h_{Paillier}(n, c) \doteq w$ is the solution. We can then make the following:

Assumption 1 *$\mathcal{H}_{Paillier}$ is 2-harder than $\mathcal{G}_{dlog}$.*

To discuss why this might be a reasonable assumption, note that no method is known to break one-way security of Paillier encryption other than factoring the modulus n. Furthermore, state of the art is (and has been for several years) that discrete log and factoring are of similar complexity for moduli of the same size. Moreover, with the current best known attacks (based on the number field sieve), doubling the modulus length has a dramatic effect on the expected time to solve the problem. Indeed, this is the reason why 1024-bit moduli are currently considered secure, even though 512-bit moduli can be broken in practice. It would therefore be very surprising, if it turned out to be possible to factor $2k$-bit numbers using only the time we need to find k-bit discrete logs. Note that if we had chosen a constant larger than 2 in Assumption 1, the assumption would be weaker, but all our results would remain essentially the same. We could even have used a polynomial f of degree ≥ 1, but then our compilation would be less efficient.

Definition 1 calls for an algorithm that completely breaks $\mathcal{G}$, and we will need this for technical reasons in the following. This makes Assumption 1 stronger than if we had only asked for one that breaks $\mathcal{G}$ in the ordinary sense. However, in the concrete case based on discrete logs, this makes no difference, as far as current state of the art is concerned: The best known attack on the discrete logarithm problem modulo p is the index calculus algorithm which works for all prime moduli, and has complexity that only depends on the size of the modulus. Furthermore, the discrete-log problem is random self-reducible and hence an algorithm solving a random instance modulo p with probability ϵ can solve any *fixed* instance modulo p with the same probability. In other words, the best known attack on the discrete log problem does in fact break it completely in our sense (albeit in superpolynomial time, of course).

2.2 Σ-protocols

Consider the following protocol (adapted from [4]), which we will call $\mathcal{P}_{eqdlog}$:

Prover P and Verifier V get as common input $x \doteq (p, p', g_1, g_2, h_1, h_2)$, where p, p' are prime, p' is k-bit long, $p = 2p' + 1$, $g_1 \in Z_p^*$ has order p', $g_2, h_1, h_2 \in \langle g_1 \rangle$ and $h_1 = g_1^w \bmod p, h_2 = g_2^w \bmod p$, for some $w \in Z_{p'}$. P gets w as private input.

1. P chooses a random $3k$-bit integer r and sends $a \doteq (a_1, a_2)$ to V, where $a_1 \doteq g_1^r \bmod p, a_2 \doteq g_2^r \bmod p$;
2. V chooses e at random in $Z_{p'}$ and sends it to P;
3. P sends $z \doteq r + ew$ to V who checks that $g_1^z = a_1 h_1^e \bmod p, g_2^z = a_2 h_2^e \bmod p$.

Define the relation R_{dlog} as the set of pairs (x, w) as specified above, and $L_{R_{dlog}} \doteq \{x | \ \exists w : (x, w) \in R_{dlog}\}$. It is easy to see that the protocol above is an interactive proof system for membership in $L_{R_{dlog}}$, that is, it proves to the verifier that $\log_{g_1}(h_1) = \log_{g_2}(h_2)$.

In general, we define a Σ-protocol [5] for a relation R to be an interactive proof systems $\mathcal{P}$ for $L_R \doteq \{x | \ \exists w : (x, w) \in R\}$ with conversations of the form (a, e, z) and with the following additional properties:

Relaxed Special Soundness: Consider an input $x \notin L_R$, and any a. We say that a value of e is *good* if there exists z such that $x, (a, e, z)$ would be accepted by the verifier. The requirement now is that for any pair $x \notin L_R, a$, at most one good e exists.

Special Honest-Verifier Zero-Knowledge: There exists a probabilistic polynomial time simulator which on input x, e outputs a conversation (a, e, z) with distribution statistically indistinguishable from conversations between P and V, for the given statement $x \in L_R$ and challenge e.

Usually, one considers Σ-protocols for R, which have the standard Special Soundness property, namely that from $x \in L_R$ and accepting conversations $(a, e, z), (a, e', z')$ where $e \neq e'$, we can efficiently compute w such that $(x, w) \in R$. This clearly implies Relaxed Special Soundness, which is all we will need here.

The properties are straightforward to verify for the example protocol $\mathcal{P}_{eqdlog}$. In addition, $\mathcal{P}_{eqdlog}$ is an example of what we call a *Σ-protocol with linear answer*:

Definition 2. *A* Σ-protocol with linear answer *is a* Σ*-protocol where the prover's final message* z *is a sequence of integers,* $z = (z_1, \ldots, z_m)$*, where* $z_j = u_j + v_j e$*, and where* u_j, v_j *are integers that can be computed efficiently from* x*,* P*'s random coins and his private input* w*.*

For a relation R to be useful, it is typically necessary that one can efficiently generate pairs $(x, w) \in R$ from a security parameter 1^k. We say that x is a k-instance, and we will assume that R comes with a polynomial ℓ_x such that k-instances have length $\ell_x(k)$.

Finally, we point out a consequence of Relaxed Special Soundness which will be important in the following: Let us consider any probabilistic polynomial-time algorithm $G_{\mathcal{P}}$ that, given a security parameter 1^k, generates a pair (x, a) where x has length $\ell_x(k)$. This defines a problem generator $\mathcal{G}_{\mathcal{P}} = \langle G_{\mathcal{P}}, g_{\mathcal{P}} \rangle$ in the sense of Section 2.1, where (x, a) is the problem instance and the solution function $g_{\mathcal{P}}$ is defined as follows: If $x \notin L_R$ and there exists a good e for (x, a), this e-value is the solution (which is unique by relaxed special soundness). These are the interesting instances. In all other cases (*i.e.,* if $x \in L_R$ or if there is no good e for (x, a)), we define the solution to be $g_{\mathcal{P}}(x, a) \doteq 0^k$ (just to ensure that there is an answer for any instance). We call any such problem generator $\mathcal{G}_{\mathcal{P}}$ a *fake-proof generator* for $\mathcal{P}$.

For the example protocol $\mathcal{P}_{eqdlog}$, it is straightforward to verify that we can find the solution to any instance (x, a) by computing a constant number of discrete logarithms mod p. Therefore, any fake-proof generator for $\mathcal{P}_{eqdlog}$ is as easy as $\mathcal{G}_{dlog}$, and so by Lemma 1, we get

Proposition 1. *Under Assumption 1,* $\mathcal{H}_{Paillier}$ *is 2-harder than any fake-proof generator for* $\mathcal{P}_{eqdlog}$*.*

2.3 Homomorphic Encryption

A public-key cryptosystem is as usual defined by algorithms E, D for encryption and decryption and a key generation algorithm KG. The key generation receives 1^k as input and outputs a pair of private and public key (sk, pk). We will consider systems where plaintexts are integers from some interval $[0, n-1]$ where n can be computed from pk. Given plaintext a and random coins r, the ciphertext is $E_{pk}(a; r)$, and we require, of course, that $a = D_{sk}(E_{pk}(a; r))$.

We will be looking at systems that are *homomorphic*, in the following sense: the set of ciphertexts is an Abelian group, where the group operation is easy to compute given the public key. Furthermore, for any a, b, r_a, r_b it holds that $E_{pk}(a; r_a) \cdot E_{pk}(b; r_b) = E_{pk}((a+b) \bmod n; s)$ for some s. We will assume throughout that n is a k-bit number. Note that by multiplying $E_{pk}(a; r)$ by a random encryption of 0, one obtains a random and independently distributed encryption of a; we denote such operation with $\mathsf{randomize}(E_{pk}(a; r))$.

A typical example of homomorphic encryption is Paillier's cryptosystem, where pk is a k-bit RSA modulus n, and sk is the factorization of n. Here, $E_{pk}(a; r) \doteq (1 + n)^a r^n \bmod n^2$, where r is uniformly chosen in Z_n^*.

2.4 The Registered Public-Key Model

Below we briefly review the registered public-key model (introduced in [1]), focusing on the aspects that we will need in the following. We refer the reader to [1] for the original description of the model and its relation to other setup assumptions (*e.g.,* the common random string model).

Let $KS(1^k)$ (for *Key Setup*) be a probabilistic polynomial-time algorithm which, on input a security parameter 1^k, outputs a private/public key pair. We write $KS(1^k; r)$ to denote the execution of KS using r as random coins.

The registered public-key model [1] features a trusted functionality F_{reg}^{KS}, which the parties can invoke to register their key pairs and to retrieve other parties' public keys. Key registration takes place by having the registrant privately sending F_{reg}^{KS} the random coins r that she used to create her key pair. F_{reg}^{KS} will then run $KS(1^k; r)$, store the resulting public key along with the identity of the registrant, and later give the public key to anyone who asks for it. Note that this in particular means that to register a public key one needs to know the corresponding private key. Note also that one need not have registered a public key of his own to ask F_{reg}^{KS} for somebody else's public key.

2.5 Non-Interactive Zero-Knowledge with Key Setup

Below we present a stand-alone definition of *Non-Interactive Zero-Knowledge* in the registered public-key model.[1]

Let $KS(1^k)$ be the key setup for the key-registration functionality F_{reg}^{KS}, and let R be a relation for which one can efficiently generate pairs $(x, w) \in R$ from a security parameter 1^k. A non-interactive system for R with key setup KS is a pair of efficient algorithms (P, V), where:

- $P(1^k, x, w, pk_V)$ is a probabilistic algorithm run by the prover. It takes as input a k-instance x and w such that $(x, w) \in R$, along with the verifier's public key pk_V, which the prover obtains from F_{reg}^{KS}. It outputs a string π as a non-interactive zero-knowledge proof that $x \in L_R$;
- $V(1^k, x, \pi, sk_V)$ is a deterministic 0/1-valued algorithm run by the verifier, satisfying the following *correctness* property: for all k-instances x and w such that $(x, w) \in R$, it holds that:

  $$\Pr[V(1^k, x, \pi, sk_V) = 1 \mid \ (sk_V, pk_V) \xleftarrow{r} KS(1^k); \pi \xleftarrow{r} P(1^k, x, w, pk_V)] = 1$$

  where the probability is over the random coins of KS and P;

The system is *zero-knowledge* if there exists a probabilistic polynomial-time algorithm M, such that for all k-instances x and w such that $(x, w) \in R$, the

[1] We only consider the setting where the key setup is required just for the verifier, as that is all we need in this paper. Adapting the definition to the case in which provers also have private/public key pair is straightforward; we omit the details.

following two ensembles are indistinguishable:

Verifier's Key Pair, Real Proof:

$\{(sk_V, pk_V, \pi) \mid (sk_V, pk_V) \xleftarrow{r} KS(1^k); \pi \xleftarrow{r} P(1^k, x, w, pk_V)\}$

Verifier's Key Pair, Simulated Proof:

$\{(sk_V, pk_V, \pi) \mid (sk_V, pk_V) \xleftarrow{r} KS(1^k); \pi \xleftarrow{r} M(1^k, x, pk_V, sk_V)\}$

As usual, depending on the quality of the indistinguishability of the above ensembles, one obtains computational, statistical or perfect zero-knowledge.

To define soundness, we consider a probabilistic polynomial-time adversary $\tilde{P}$ who plays the following game:

– Execute $(sk_V, pk_V) \xleftarrow{r} KS(1^k)$ and give pk_V to $\tilde{P}$.
– Repeat until $\tilde{P}$ stops: $\tilde{P}$ outputs x, π and receives $V(1^k, x, \pi, sk_V)$.

We say that $\tilde{P}$ wins if he produces at least one x, π that V accepts, where $x \notin L_R$. The protocol is *sound* if any $\tilde{P}$ wins with probability negligible in k. We say that the system is sound for a particular number of proofs $m(k)$ if the game always stops after at most $m(k)$ proofs are generated.

3 A Compilation Technique

In this section, we assume we are given a relation R and a Σ-protocol $\mathcal{P}$ for R with linear answer. When running the protocol on input (x, w), where x is a k-instance, we let $\ell_x(k)$ be the bit-length of x, $\ell_e(k)$ be the bit-length of the verifier's challenge, and $\ell_z(k)$ be the maximal bit-length of a component in the prover's answer z *i.e.*, $z = (z_1, \dots, z_m)$ and $\ell_z(k) \doteq \max(len(z_1), \dots, len(z_m))$. We also use a homomorphic cryptosystem with key generation algorithm KG.

Our compilation technique works in the registered public-key model of [1] (*cf.* also Section 2.4). Specifically, we assume that each player acting as verifier has initially registered a private/public key pair with the trusted functionality F_{reg}^{KS}, using the following key setup algorithm:

$KS(1^k)$ (Key setup for the Verifier):
Set $(sk, pk) \xleftarrow{r} KG(1^{k'})$ where we choose $k' \doteq \max(f(k)k, \ell_z(k) + 1)$, and where $f(k)$ is a polynomial specified in Theorem 2 below. Choose a challenge e as V would do in the given Σ-protocol (that is, e will be a $\ell_e(k)$-bit string), and set c to be a random (homomorphic) encryption of e under pk. The public key is now (pk, c) and the private key is (sk, e).

In Section 6, we discuss how our key setup functionality F_{reg}^{KS} can be implemented efficiently in a standard PKI setting.

Note that the algorithm $KS(1^k)$ for the verifier's key setup can also be thought of as defining a problem generator, where (pk, c) is the problem instance, and e is the solution. We will call this problem generator $\mathcal{H}_{KG}$ in the following. It will be identical to $\mathcal{H}_{Paillier}$ if we use Paillier encryption.

To understand the compilation technique itself, note that because the Σ-protocol is with linear answer, it is possible to execute the prover's side of the protocol given only an encryption of the challenge e. Namely, the prover starts by computing his first message a. Then, if the answer z is supposed to contain $z_j = u_j + v_j e$, the prover will be able (by linearity) to derive the values of u_j, v_j from x, his private input w and the random coins used to create a. At this point, the prover can compute $E_{pk}(z_j)$ as $E_{pk}(u_j) \cdot c^{v_j}$. This can be decrypted and then checked as usual by V.

Now, soundness of any Σ-protocol is based on the fact that a cheating prover has to generate the first message a without knowing what the challenge is. Since, in this case, the prover is only given an encryption of the challenge, we might hope that soundness would still hold. More specifically, if the prover can, for a false statement x, come up with a first message a, and encrypted responses that the verifier would accept, then relaxed special soundness implies that x, a uniquely determines the challenge e that the verifier encrypted. If the complexity of finding e from x, a is much smaller than the complexity of breaking the verifier's cryptosystem, this gives a contradiction, as formalized below. On the other hand, zero-knowledge simulation is easy if the challenge is known to V, and the key setup exactly guarantees that V knows the challenge.

A more detailed description of the compiled protocol follows. Our construction is designed to give proofs for instances x of length up to $\ell_x(k)$. It is in general understood that the verifier will reject immediately if x is longer than $\ell_x(k)$ or if the proof is in any other way obviously malformed.

Protocol $\mathsf{compile}(\mathcal{P})$

1. Given a k-instance x, w to prove, P gets V's public key (pk, c) from F_{reg}^{KS} and computes the first message a in a proof according to $\mathcal{P}$. Let the final message z be of the form $(u_1 + v_1 e, \dots, u_m + v_m e)$; then, for $i = 1, \dots, m$, P computes $c_i \overset{r}{\leftarrow} \mathsf{randomize}(E_{pk}(u_j) \cdot c^{v_j})$. P sends x, π to V, where $\pi \doteq (a, (c_1, \dots, c_m))$.
2. On input x and a proof $\pi \doteq (a, (c_1, \dots, c_m))$, V sets $z'_i \leftarrow D_{sk}(c_i)$, and then verifies that $x, (a, e, (z'_1, \dots, z'_m))$ would be accepted by the verifier of protocol $\mathcal{P}$, and accepts or rejects accordingly.

Theorem 1. $\mathsf{compile}(\mathcal{P})$ *is complete and statistical zero-knowledge (in the registered public-key model).*

Proof. Completeness is clear by inspection. In particular, $D_{sk}(c_i)$ equals the correct value $z_i \doteq u_i + v_i e$, since the fact that $k' > \ell_z(k)$ ensures that $z_i < n$.

As for zero-knowledge, the simulator M will as usual interact with V and attempt to emulate the view V would see in real life. In particular, M will receive the string V sends initially (namely, the random coins r intended for F_{reg}^{KS}). This allows M to generate V's private key, and in particular the e-value inside c. Now, to simulate a proof for $x \in L_R$, M will use the special honest-verifier simulator for $\mathcal{P}$ on input x, e to generate $(a, e, z) = (a, e, (z_1, \dots, z_m))$. It then outputs $x, (a, (E_{pk}(z_1), \dots, E_{pk}(z_m)))$. The only difference between this simulation and real proofs is that the values $a, z_1, \dots, z_m$ are generated by the prover in $\mathcal{P}$ in real proofs, while in M's output they are simulated. The theorem now follows from special honest-verifier zero-knowledge of $\mathcal{P}$. $\square$

Theorem 2. *Let $\mathcal{P}$ be a Σ-protocol with linear answer, and $\mathcal{H}_{KG}$ be the problem generator associated with the key setup for the verifier. Assume that $\mathcal{H}_{KG}$ is f-harder than any fake-proof generator $\mathcal{G}_{\mathcal{P}}$ for $\mathcal{P}$, and that the verifier's public key for the homomorphic encryption scheme is generated with security parameter $1^{k'}$, where $k' \doteq max(f(k)k,\ \ell_z(k)+1)$. Then* compile($\mathcal{P}$) *is sound for provers generating $O(\log k)$ proofs.*

Proof. Assume we have a probabilistic polynomial-time cheating prover $\tilde{P}$ contradicting the conclusion of the theorem. At a high level, our proof will proceed as follows: first, we describe how to use $\tilde{P}$ to obtain a fake-proof generator $\tilde{\mathcal{G}}_{\mathcal{P}} = \langle \tilde{G}_{\mathcal{P}}, g_{\mathcal{P}} \rangle$ for $\mathcal{P}$; then, using $\tilde{P}$ and any algorithm A that completely breaks $\tilde{\mathcal{G}}_{\mathcal{P}}$, we will show how to construct an algorithm A' breaking $\mathcal{H}_{KG}$ on instances of size $k' \geq f(k)k$, in time comparable to A's. This will contradict the assumption that $\mathcal{H}_{KG}$ is f-harder than any fake-proof generator for $\mathcal{P}$.

Consider the algorithm $\tilde{G}_{\mathcal{P}}$ which, on input 1^k, starts by generating a public key (pk, c) for the verifier according to the protocol (*i.e.*, (pk, c) was produced by $KS(1^{k'})$). Then, $\tilde{G}_{\mathcal{P}}$ runs $\tilde{P}$ on (pk, c), and whenever $\tilde{P}$ outputs a statement/proof pair, $\tilde{G}_{\mathcal{P}}$ replies with a random bit to represent the verifier's reaction to each proof. Once $\tilde{P}$ halts, $\tilde{G}_{\mathcal{P}}$ chooses uniformly one of the statement/proof pairs generated by $\tilde{P}$ (it will be of the form x,$(a$,$(c_1, \ldots, c_m)))$, and outputs (x, a).

Note that with probability $1/poly(k)$, all the bits that $\tilde{G}_{\mathcal{P}}$ sends to $\tilde{P}$ are identical to what the verifier would have sent. Hence, the fact that $\tilde{P}$ is a successful cheating prover implies that, with non-negligible probability, one of the statement/proof pairs $x, (a, (c_1, \ldots, c_m))$ generated by $\tilde{P}$ is such that $x \notin L_R$, yet the verifier would accept. Given that there is such a proof, there is at least a $1/(\log k)$ probability that $\tilde{G}_{\mathcal{P}}$ chooses this proof to generate its output. In conclusion, with overall non-negligible probability, $\tilde{G}_{\mathcal{P}}$ outputs $x \notin L_R, a$ for which exactly one good e exists. This value of e must be identical to the plaintext inside c since the verifier would accept the corresponding proof.

Algorithm $\tilde{G}_{\mathcal{P}}$ defines a fake-proof generator $\tilde{\mathcal{G}}_{\mathcal{P}} = \langle \tilde{G}_{\mathcal{P}}, g_{\mathcal{P}} \rangle$ for $\mathcal{P}$ (where, as in Section 2.2, $g_{\mathcal{P}}(x, a)$ is the good e-value if one exists and $x \notin L_R$, and 0^k otherwise). Hence, the assumption that $\mathcal{H}_{KG}$ is f-harder than any fake-proof generator for $\mathcal{P}$ implies in particular that $\mathcal{H}_{KG}$ is f-harder than $\tilde{\mathcal{G}}_{\mathcal{P}}$.

Let A be a probabilistic algorithm that breaks $\tilde{\mathcal{G}}_{\mathcal{P}}$ completely in time $T(k)$, and consider the following algorithm A' to break $\mathcal{H}_{KG}$. On input a k'-instance (pk, c) for $\mathcal{H}_{KG}$ (*i.e.*, (pk, c) was produced by $KS(1^{k'})$) A' invokes $\tilde{P}$ on (pk, c) and interacts with it according to the exact same strategy that we described above for $\tilde{G}_{\mathcal{P}}$. At the end of such interaction, A' will obtain a pair (x, a): at this point, A' runs A on (x, a), and outputs the value e returned by A.

By the above analysis of $\tilde{G}_{\mathcal{P}}$ and the fact that A breaks $\tilde{\mathcal{G}}_{\mathcal{P}}$ completely, we see that A' returns the plaintext encrypted inside c with non-negligible probability. Since A' runs in time $T(k) + poly(k)$ and $k' \geq f(k)k$, this contradicts the assumption that $\mathcal{H}_{KG}$ is f-harder than $\tilde{\mathcal{G}}_{\mathcal{P}}$. $\square$

For the example protocol $\mathcal{P}_{eqdlog}$, the above theorem and Proposition 1 imply the following:

Corollary 1. *Suppose we construct* $\mathsf{compile}(\mathcal{P}_{eqdlog})$ *using Paillier encryption with security parameter* $1^{k'}$*, where* $k' \doteq \max(2k, \ \ell_z(k)+1)$*. Then, under Assumption 1,* $\mathsf{compile}(\mathcal{P}_{eqdlog})$ *is sound for provers generating* $O(\log k)$ *proofs. Moreover, its communication and computational complexity are a constant factor times those of* $\mathcal{P}_{eqdlog}$*.*

While the restriction to a logarithmic number of proofs may seem like a serious one, there are in fact many applications where this result is good enough. The point is that our reduction only fails for polynomially-many proofs because we assume that the prover learns whether the verifier accepts each individual proof. However, when a zero-knowledge protocol is used as a tool in a larger construction, the prover often does not get this information, and thus in such cases, it is enough that soundness holds for a single proof. The application to threshold RSA in the next section is an example of this.

Moreover, we believe that $\mathsf{compile}(\mathcal{P}_{eqdlog})$ is in fact sound, even for an arbitrary polynomial number of proofs. We can show this under a stronger non-standard assumption: we report the details in Appendix A.

4 Threshold RSA

Our technique can be used in most known threshold RSA- or discrete-log-based cryptosystems to obtain efficient solutions not relying on random oracles. As a concrete example, we consider here Shoup's threshold RSA protocol [17].

In this construction, a trusted dealer generates an RSA modulus $N = pq$, where $p = 2p'+1, q = 2q'+1$ and p', q' are k-bit primes. In addition, the dealer publishes an element $v \in Z_N^*$ of order $p'q'$, and sets up a secret sharing of the private exponent. Each server S_i in the protocol privately receives a share s_i (which is a number modulo $p'q'$). Finally, the dealer publishes the value $v_i \doteq v^{s_i} \bmod N$ for each server.

When the system is operational, a client may send an input α to be signed to all servers. Each server S_i in the protocol produces an element β_i which is guaranteed to be in the subgroup of Z_N^* of order $p'q'$ (because it is a square of another element). Server S_i then sends β_i to the client, claiming that $\beta_i = \alpha^{s_i} \bmod N$. Assuming that the majority of the servers are honest, the client can reconstruct the desired signature, as long as he does not accept any incorrect β_i's. Each server must therefore prove to the client that β_i was correctly formed.

The following Σ-protocol $\mathcal{P}_{dlmodN}$ can be used as the basis for a solution:

1. S_i chooses a random $4k$-bit integer r and sends $a \doteq (a_1, a_2)$ to V, where $a_1 \doteq v^r \bmod N, a_2 \doteq \alpha^r \bmod N$.
2. The verifier chooses a random $(k-1)$-bit string e and sends it to P.
3. S_i sends $z \doteq r + es_i$ to the verifier who checks that $v^z = a_1 v_i^e \bmod N, \alpha^z = a_2 \beta_i^e \bmod N$.

Assuming that N, v are generated by the trusted dealer as described, it follows from the arguments given in [17] that this is a Σ-protocol for proving that $\log_v(v_i) = \log_\alpha(\beta_i)$. Indeed, the non-interactive solution proposed in [17] is simply the Fiat-Shamir heuristic applied to this protocol.

We propose to apply instead our compilation technique based on Paillier encryption to get a non-interactive solution. This leads to:

Theorem 3. *Under Assumption 1, there exists a non-interactive threshold RSA scheme, secure in the registered public-key model. Its communication and computational complexity are the same as in Shoup's scheme, up to a constant factor.*

Proof. The protocol given above has the right properties for applying the compilation technique, the only exception being a small technical issue with soundness: the protocol has relaxed special soundness only for inputs where N, v are correctly formed, while our definition requires it for all inputs. However, we can simply instruct the verifier to reject all inputs not containing the N, v generated by the dealer. This will force a cheating prover to only use inputs for which relaxed special soundness holds, and the proof of Theorem 2 then goes through in the same way as before.

To apply Theorem 2, we need to show that $\mathcal{H}_{Paillier}$ is $f(k)$-harder (for some $f(k)$) than any fake-proof generator $\mathcal{G}_{dlmodN} = \langle G_{dlmodN}, g_{dlmodN} \rangle$ for $\mathcal{P}_{dlmodN}$. To this end, observe that when we argue soundness, we may assume that the factors p, q of N are known, since soundness is based only on security of the (independently chosen) Paillier public key, specified by the verifier's key pair. Now, instances for a fake-proof generator $\mathcal{G}_{dlmodN}$ have the form $(x, a) \doteq ((N, v, v_i, \alpha, \beta_i), (a_1, a_2))$, whereas the solution $g_{dlmodN}(x, a)$ typically is the only e-value that S_i can answer (unless either there is no such e-value, or the theorem x is true, in which cases $g_{dlmodN}(x, a) \doteq 0^k$). Since p, q are known, we can reduce everything modulo p and q and the Chinese remainder theorem now implies that we can find the solution by computing a constant number of discrete logarithms mod p and q. Consequently, any fake-proof generator for $\mathcal{P}_{dlmodN}$ is as easy as $\mathcal{G}_{dlog}$; therefore Assumption 1 implies that $\mathsf{compile}(\mathcal{P}_{dlmodN})$ is sound against provers giving $O(\log k)$ proofs (though, as we will see below, we only need soundness for provers giving a single proof).

To prove that the RSA protocol is secure, we must first show that an adversary corrupting at most half the servers learns nothing from the protocol, except for the RSA signatures that the protocol is supposed to produce. This follows from zero-knowledge of $\mathsf{compile}(\mathcal{P}_{dlmodN})$ and the simulator given in [17].

Second, we must show that no probabilistic polynomial-time adversary can make an honest client fail to output a correct RSA signature (even on input messages chosen by the adversary). We will show that existence of an adversary Adv doing this with non-negligible probability contradicts soundness of $\mathsf{compile}(\mathcal{P}_{dlmodN})$, namely we construct from Adv a prover that cheats the client on a single proof with non-negligible probability.

For this, we will execute the dealer's algorithm to set up the RSA key and give shares of the private key to the adversary for those servers he wants to corrupt. We also give him the public key of the client we want to attack. Assume *Adv* chooses a maximum of $i_{\max}$ input messages for the client before halting. We pick i at random in $[1, i_{\max}]$ and hope that the i-th message is the first where *Adv* is successful in cheating the client. Since $i_{\max}$ is polynomial in k, our guess is correct with $1/poly(k)$ probability. Assuming our guess is correct, we can perfectly simulate what *Adv* sees for any previous message m_j, $j < i$: for the actions of honest servers, we can simply follow the protocol (as we know the private RSA key and all its shares); as for the client, for $j < i$ he will just output a correct RSA signature on m_j, which we can also compute.

Since (assuming a correct guess of i) we can perfectly simulate *Adv*'s view up to message m_i, there is a non-negligible probability that *Adv* successfully cheats the client when he tries to get a signature on m_i. But for this to happen, *Adv* must fool the client into accepting an incorrect share, which can only occur if *Adv* produced (for at least one of the corrupt servers) an acceptable proof for an incorrect statement. Thus, we choose at random one of the corrupt servers and output its statement and proof. This is clearly a successful cheating prover. □

5 The OR-Construction and NIZKs for NP

5.1 Closure under OR-Construction

A construction that is widely used in designing efficient Σ-protocols is the so-called OR-construction [8]. Given Σ-protocols Σ_l and Σ_r for relations R_l and R_r, the OR-construction yields a Σ-protocol Σ_{OR} for the following relation R_{OR}:

$$((x_l, x_r), (w_l, w_r)) \in R_{OR} \Leftrightarrow ((x_l, w_l) \in R_l \vee (x_r, w_r) \in R_r).$$

The OR-construction is based on executing the two protocols for relations R_l, R_r in parallel, where the prover derives the two challenges from a single value chosen by the verifier. In our case, we do all computations on challenges over the integers, which means that some details of the standard construction have to be modified slightly; this is covered in Appendix B.

An attractive feature of the compilation technique proposed in Section 3 is that if it is applicable to both Σ_l and Σ_r, then it is also applicable to the composed protocol Σ_{OR}. In other words:

Theorem 4. *The class of Σ-protocols that can be made non-interactive using our homomorphic-encryption-based technique is closed under OR-construction.*

Proof. Let Σ_l and Σ_r be Σ-protocols with linear answer. The theorem amounts to proving that the Σ-protocol Σ_{OR} resulting from the OR-construction also features a "linear answer," and so we can apply the compiler from Section 3. Now, valid conversations of Σ_{OR} (*cf.* Appendix B) have the form:

$$((a_l, a_r), e, (e_l, z_l, e_r, z_r)),$$

where (e_l, e_r) is a "split" for e, that is, $e = e_l - e_r$ and either e_l or e_r was chosen randomly by the prover when preparing (a_l, a_r). Hence, e_l and e_r are clearly linear; moreover, since both Σ_l and Σ_r have linear answer, z_l and z_r are also linear, and the theorem follows. □

5.2 Non-Interactive Bit Commitments

We now describe a non-interactive bit-commitment scheme for the registered public-key model, along with non-interactive protocols to prove boolean relations among committed bits.

Consider the Σ-protocol $\mathcal{P}_{eqdlog}$ for equality of discrete logarithms described in Section 2.2. Applying the OR-construction to two instances of $\mathcal{P}_{eqdlog}$ yields a Σ-protocol $\mathcal{P}_{1out2}$ for proving that one out of two pairs of discrete logarithms is equal. In other words, $\mathcal{P}_{1out2}$ is a proof system for statements of the form $x \doteq (p, p', g_1, g_2^0, g_2^1, h_1, h_2)$, where p, p' are prime, $p = 2p' + 1$, $g_1, g_2^0, g_2^1 \in \mathbb{Z}_p^*$ have order p', $h_1 = g_1^w \bmod p$ (for some $w \in \mathbb{Z}_{p'}$) and either $h_2 = (g_2^0)^w \bmod p$ or $h_2 = (g_2^1)^w \bmod p$.

To commit to a bit b, the prover picks p, p', g_1, g_2^0, g_2^1 as described above,[2] randomly selects $w \in \mathbb{Z}_{p'}$ and computes $h_1 = g_1^w \bmod p$, $h_2 = (g_2^b)^w \bmod p$. At this point, the prover uses $\mathsf{compile}(\mathcal{P}_{1out2})$ to prove (non-interactively) that the statement $x \doteq (p, p', g_1, g_2^0, g_2^1, h_1, h_2)$ is well-formed. The commitment then consists of x along with such NIZK, though in the following we will often refer to x by itself as the commitment to keep the discussion simpler.

To open the commitment to b, it suffices to show that $\log_{g_1} h_1 = \log_{g_2^b} h_2$, which the prover can do non-interactively via the protocol $\mathsf{compile}(\mathcal{P}_{eqdlog})$.

Now, suppose that we want to show that three bits b_1, b_2, b_f (hidden within commitments x_1, x_2, x_f, respectively) satisfy $b_f = f(b_1, b_2)$, for some binary boolean function f. Proving such relation amounts to prove that (x_1, x_2, x_f) can be opened either to $(0, 0, f(0, 0))$, or to $(0, 1, f(0, 1))$, or to $(1, 0, f(1, 0))$, or to $(1, 1, f(1, 1))$. But this is just the disjunction of statements that can each be proven using three instances of $\mathcal{P}_{eqdlog}$; hence, applying the OR-construction we get a Σ-protocol Σ_f that can be made non-interactive as described in Section 3.

5.3 NIZK for Circuit Satisfiability

The discrete-logarithm-based non-interactive bit-commitment scheme from Section 5.2 can be used, in conjunction with the approach of [6], to obtain efficient non-interactive zero-knowledge arguments for Circuit Satisfiability, and hence for any NP language.

To show that a given circuit is satisfiable, the prover P commits to his satisfying assignment and to all intermediate bits resulting form the computation of the circuit, and sends all these non-interactive bit-commitments to the verifier V. Additionally, P non-interactively opens the output bit to 1, and prove non-interactively to the verifier that the commitments to the inputs and the output of each gate of the circuit are consistent.

Upon receiving such non-interactive proof, V checks that all the commitments are well-formed, that the output of the circuit actually opens to 1, and that the proof of consistency of each gate is correct, and if so, V accepts P's proof.

[2] As a matter of efficiency, we notice that, when committing to many bits, the prover can safely reuse the values p, p', g_1, g_2^0 and g_2^1.

Notice that the length of such non-interactive proof is proportional to the circuit's size and to the security parameter 1^k, and is thus "linear" in the sense of the "Linear Zero-Knowledge" of [6], whereas previous constructions [14] in the common random string model are quadratic in this regard, even under specific number-theoretic assumptions.

6 Implementing the Key Setup

The compilation technique of Section 3 works in the registered public-key setting. In this model, each verifier V registers her public key by sending the random coins used to generate her private key/public key pair to a trusted functionality. This is exploited in the proof of Theorem 1 to enable the simulator M to derive the private key of the verifier, and to ensure the validity of the public key.

Of course, such a functionality can always be implemented using a more standard PKI with a certification authority CA, and generic zero-knowledge techniques. The verifier sends her public key to the CA and proves in zero-knowledge that she knows a set of random coins that, using the given key-generation algorithm, leads to the public key she sent.

This will be very inefficient in general. But in fact, taking a closer look at the simulation for the case where the verifier uses Paillier encryption, one can see that all that is needed is knowledge of the challenge value e and of the RSA modulus n, plus assurance that e lies in the proper interval and that n is well-formed. (Knowledge of the factorization of n, in particular, is not required.) In our case, it is enough to know that n is the product of two distinct primes and that n is relatively prime to $\phi(n)$. Hence, registration of the verifier's key pair for the key setup from Section 3 can be efficiently implemented by having V and CA engage in the following protocol:

Step 0: V sends her public key (n, c) to CA;
Step 1: V proves to CA that n is well-formed;
Step 2: V proves knowledge of the plaintext e hidden within c; and that this value e lies in the specified interval.

All the above steps can be efficiently realized leveraging known tools from the literature [18, 2, 9]. In particular, Step 1 can be carried out by first using the protocol of van de Graaf and Peralta [18], by which one can show that $n = p^i q^j$ where $p \equiv q \equiv 3 \bmod 4$ and i, j are odd. Then one can use the following folklore trick: the verifier chooses a random element in Z_n^*, and the prover proves in zero-knowledge that it has an n-th root mod n. This will always be the case if $\gcd(n, \phi(n)) = 1$, but fails with constant probability otherwise. As for Step 2, one can first use an integer commitment scheme (like the one of Damgård and Fujisaki [9]) to create a commitment Com to e, and then prove knowledge of the value committed within Com (*e.g.*, using the protocol in Section 4.1 of [9]). Then, using standard techniques, it is possible to show that the commitment Com and the ciphertext c hide the same value e. For completeness, in Appendix C we sketch a simple Σ-protocol to achieve this. Finally, Boudot's efficient proof of membership in intervals [2] allows the prover to prove that the e contained in Com lies in the required range.

Acknowledgement. We thank the anonymous referees for useful advise on improving the presentation.

References

1. B. Barak, R. Canetti, J. B. Nielsen, and R. Pass. Universally Composable Protocols with Relaxed Set-Up Assumptions. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04)*, pages 186–195. IEEE Computer Society, 2004.
2. F. Boudot. Efficient Proofs that a Commited Number Lies in an Interval. In *Advances in Cryptology—EUROCRYPT '00*, volume 1807 of *LNCS*, pages 431–444. Springer, 2000.
3. R. Canetti, O. Goldreich, S. Goldwasser, and S. Micali. Resettable zero-knowledge. In *STOC'99*, pages 235–244. ACM Press, 1999.
4. D. Chaum and T. P. Pedersen. Wallet databases with observers. In *Advances in Cryptology—CRYPTO '92*, volume Volume 740 of *LNCS*, pages 89–105. Springer, 1992.
5. R. Cramer. *Modular Design of Secure yet Practical Cryptographic Protocols.* PhD thesis, CWI and University of Amsterdam, 1996.
6. R. Cramer and I. Damgård. Linear Zero-Knowledge—A Note on Efficient Zero-Knowledge Proofs and Arguments. In *Proceedings of the* 29^{th} *Annual ACM Symposium on Theory of Computing*, pages 436–445. ACM Press, 1997.
7. R. Cramer and I. Damgård. Secret-Key Zero-Knowledge. In *Theory of Cryptography—TCC '04*, pages 223–237. Springer-Verlag, 2004. LNCS 2951.
8. R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *Advances in Cryptology—CRYPTO '94*, pages 174–187. Springer, 1994. LNCS 839.
9. I. Damgård and E. Fujisaki. A Statistically-Hiding Integer Commitment Scheme Based on Groups with Hidden Order. In *Advances in Cryptology—ASIACRYPT '02*, pages 125–142. Springer, 2002. LNCS 2501.
10. C. Dwork and M. Naor. Zaps and Their Applications. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS'00)*, pages 283–293. IEEE Computer Society, 2000.
11. A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology—Crypto'86*, volume 263 of *LNCS*, pages 186–194, Berlin, 1987. Springer.
12. S. Goldwasser and Y. Tauman Kalai. On the (In)security of the Fiat-Shamir Paradigm. In *FOCS '03*, pages 102–115. IEEE Computer Society, 2003.
13. J. Groth, R. Ostrovsky, and A. Sahai. Perfect Non-Interactive Zero Knowledge for NP. http://eprint.iacr.org/2005/290, 2005.
14. Joe Kilian and Erez Petrank. An Efficient Non-interactive Zero-Knowledge Proof System for NP with General Assumptions. *J. Cryptology*, 11(1):1–27, 1998.
15. P. Paillier. Public Key Cryptosystems Based on Composite Degree Rediduosity Classes. In *Advances in Cryptology—EUROCRYPT '99*, pages 223–238. Springer, 1999. LNCS 1592.
16. C. Schnorr. Efficient Signature Generation by Smart Cards. *Journal of Cryptology*, 4(3):161–174, 1991.

17. V. Shoup. Practical Threshold Signatures. In *Advances in Cryptology—EUROCRYPT '00*, pages 207–220. Springer, 2000. LNCS 1807.
18. J. van de Graaf and R. Peralta. A Simple and Secure Way to Show Validity of Your Public Key. In *Advances in Cryptology—CRYPTO '87*, volume 293 of *LNCS*, pages 128–134. Springer, 1988.

A Unbounded Soundness of $\mathsf{compile}(\mathcal{P}_{eqdlog})$

Below we sketch an argument showing soundness of $\mathsf{compile}(\mathcal{P}_{eqdlog})$ for provers generating any polynomial number of NIZKs, assuming Paillier cryptosystem is used for the homomorphic encryption. Throughout, all exponentiations are meant modulo p, unless noted otherwise.

Recall that valid statements for the protocol $\mathcal{P}_{eqdlog}$ have the form $x \doteq (p, p', g_1, g_2, g_1^w, g_2^w)$, where w is the secret input to the prover. Using w, an honest prover computes his proof as $\sigma \doteq ((g_1^r, g_2^r), E_n(r) \cdot c^w \bmod n^2)$, where c is the encrypted challenge and n is the verifier's modulus.

Our argument works in a "generic model" for the homomorphic encryption scheme: namely, we assume that whenever the prover outputs a proof $\sigma \doteq ((a_1, a_2), \bar{c})$, the ciphertext $\bar{c}$ is specified as a pair of integers (u, v) such that $\bar{c} = E_n(u) \cdot c^v \bmod n^2$.

Theorem 5. *Suppose we construct* $\mathsf{compile}(\mathcal{P}_{eqdlog})$ *using Paillier encryption. Then under Assumption 1,* $\mathsf{compile}(\mathcal{P}_{eqdlog})$ *is unboundedly sound for provers using the homomorphic properties of Paillier encryption in a black-box fashion.*

Proof. Let A be an algorithm completely breaking k-instances of $\mathcal{G}_{dlog}$ in time $T(k)$, and assume we have a cheating prover $\tilde{P}$ contradicting the conclusion of the theorem. We show how to use A and $\tilde{P}$ to construct an algorithm A' that breaks $2k$-instances of $\mathcal{H}_{Paillier}$ in time $O(T(k) + poly(k))$, contradicting Assumption 1.

On input (n, c), A' starts by executing $\tilde{P}$ on the same values n, c. During its execution, $\tilde{P}$ produces several statement/proof pairs x, σ to which A' ought to reply with a bit representing the verifier's reaction. Let $x \doteq (p, p', g_1, g_2, h_1, h_2)$ and $\sigma \doteq ((a_1, a_2), (u, v))$; then A' replies with 1 if and only if the following relations hold:

$$h_1 = g_1^v \bmod p, \quad h_2 = g_2^v \bmod p, \quad a_1 = g_1^u \bmod p, \quad a_2 = g_2^u \bmod p \qquad (\sharp)$$

When $\tilde{P}$ stops running, A' picks at random a statement/proof pair x, σ among those produced by $\tilde{P}$. Then, calling A twice, A' can compute $w_1 \doteq \log_{g_1} h_1$ and $w_2 \doteq \log_{g_2} h_2$, thus being able to decide whether x is a valid statement or not. In either case, A' can recover e from σ with either one or two more calls to A:

A-1. if x is a false statement (*i.e.,* $w_1 \neq w_2$), then A' invokes A to learn $r_1 \doteq \log_{g_1} a_1$, $r_2 = \log_{g_2} a_2$, and computes $e \doteq (r_2 - r_1)(w_1 - w_2)^{-1} \bmod p$;
A-2. if x is a valid statement (*i.e.,* $w_1 = w_2 = w$), then A' invokes A to learn $r \doteq \log_{g_1} a_1$, and computes $e \doteq (r - u)(v - w)^{-1} \bmod p$ (if $w = v$, then A' aborts).

The running time of A' is clearly $O(T(k)+poly(k))$. We now argue about its success probability. In the analysis, we use the term "funny" proof to refer to a proof σ for a true statement x that was not obtained according to the protocol, yet it passes the verifier's test. In our "generic model," given the fact that $\mathcal{P}_{eqdlog}$ admits at most one valid answer z for any given x, a, e, a funny proofs satisfies $(u,v) \neq (\log_{g_1} a_1, \log_{g_1} h_1)$, but $u + v \cdot e = \log_{g_1} a_1 + \log_{g_1} h_1 \cdot e$.

The view that $\tilde{P}$ sees within the simulation put on by A' deviates from what $\tilde{P}$ would see in a real interaction with the verifier only after $\tilde{P}$ produces a statement/proof pair x, σ for which either of the following two cases occurs:

B-1. x is a false statement, but σ passes the verifier's test (whereas according to the test ($\sharp$), A' always rejects σ in such case);
B-2. x is a true statement, but σ is a "funny" proof (notice that the test ($\sharp$) ensures that A' rejects all proofs not created according to the protocol).

Observe that since $\tilde{P}$ is a successful cheating prover, then at least one of the above cases will occur with non-negligible probability. Let i^* be the index of the first such occurrence. With $1/poly(k)$ probability, the random statement/proof pair chosen by A' will be exactly the i^*-th pair. Conditioning on such event, the simulation of the verifier's answers up to that point is perfect, and moreover:

C-1. if i^* corresponds to a false statement (case B-1. above), then the fact that σ passes the verifier's condition, along with relaxed special-soundness, implies that the value e computed by A' according to case A-1. is indeed correct;
C-2. if i^* corresponds to a "funny" proof (case B-2. above), then the fact that σ passes the verifier's condition implies that the value e computed by A' according to case A-2. is correct. (Notice that σ being a "funny" proof excludes the possibility of aborting in case A-2.)

In conclusion, with non-negligible probability, A' outputs the correct solution e to the $2k$-instance n, c in time $O(T(k)+poly(k))$, contradicting the assumption that $\mathcal{H}_{Paillier}$ is 2-harder than $\mathcal{G}_{dlog}$. $\square$

B The OR-Construction of [8]

The OR-construction [8] derives a Σ-protocol Σ_{OR} from Σ_l and Σ_r by allowing the prover to "split" the challenge $e \in [0, 2^k[$ into two parts $e_l, e_r \in [0, 2^{2k}[$ as he wishes, as long as $e_l - e_r = e$. This enables the prover to "simulate" the false part of the statement, while actually carrying out the proof for the part which is true. More in details, conversations in the OR-construction have the form:

$$((a_l, a_r), e, (e_l, z_l, e_r, z_r)),$$

where an honest prover P constructs his flows differently depending on whether P holds a valid witness w_l for x_l, or a valid w_r for x_r.

In the first case, P picks a random e_r from $[0, 2^{2k}[$ and uses the simulator for Σ_r to obtain an accepting conversation (a_r, e_r, z_r) for x_r. Then, P selects a_l

according to Σ_l and sends (a_l, a_r) to V. When P receives e, he sets $e_l \doteq e_r + e$ and computes z_l with respect to x_l, a_l, e_l and the witness w_l, according to Σ_l.

The second case is completely analogous, except that P sets $e_r \doteq e_l - e$.

As for the verification condition, V checks that (a_l, e_l, z_l), (a_r, e_r, z_r) are accepting conversations respectively for x_l and x_r, and that (e_l, e_r) is a valid "split" for e, that is, $e = e_l - e_r$.

Observe that choosing e_l and e_r to be k bits longer than e ensures that the joint distribution of (e_l, e_r) does not reveal (to the verifier) information about whether P had a valid witness for the "left" or for the "right" part of Σ^{OR}. Indeed, given any fixed value of e in $[0, 2^k[$, the statistical distance between the two marginal distributions on (e_l, e_r) induced by the experiments described below is clearly negligible in k:

"Left" distribution: randomly choose e_r from $[0, 2^{2k}[$, and set $e_l \doteq e_r + e$;
"Right" distribution: randomly choose e_l from $[0, 2^{2k}[$, and set $e_r \doteq e_l - e$.

C An Efficient Sub-protocol for the Key Setup

Let n be the verifier's modulus, e be her secret k-bit challenge, and c be a random Paillier encryption of e under n, namely $c \leftarrow (1+n)^e r^n \bmod n^2$, for some random $r \in \mathbb{Z}_n^*$. Recall that in the integer commitment scheme of [9], a commitment Com to e has the form $Com \doteq G^e H^s \bmod N$, where N is the product of two k-bit strong primes, G, H are generators of the subgroup of quadratic residues modulo N, and s is a $2k$-bit randomizer.

For binding, it is important that the verifier does not know neither the factorization of N nor the discrete log of H base G. In our setting, this can be enforced by having CA choosing G, H and N. Afterward, V can prove to CA that c and Com hide the same value via the following protocol:

1. V randomly selects $\hat{e} \in [0, 2^{3k}[$, $\hat{s} \in [0, 2^{4k}[$, $\hat{r} \in \mathbb{Z}_n^*$, and sends CA the values $\widehat{Com} \leftarrow G^{\hat{e}} H^{\hat{s}} \bmod N$ and $\hat{c} \leftarrow (1+n)^{\hat{e}} \hat{r}^n \bmod n^2$;
2. CA replies with a random $(k-1)$-bit challenge t;
3. V computes $\tilde{e} \leftarrow \hat{e} + et$ and $\tilde{s} \leftarrow \hat{s} + st$ (over the integers), and $\tilde{r} \leftarrow \hat{r} \cdot r^t \bmod n$, and sends $\tilde{e}$, $\tilde{r}$, $\tilde{s}$;
4. CA checks that $G^{\tilde{e}} H^{\tilde{s}} \stackrel{?}{=} \widehat{Com} \cdot Com^t \bmod N$ and $(1+n)^{\tilde{e}} \tilde{r}^n \stackrel{?}{=} \hat{c} \cdot c^t \bmod n^2$.

It is easy to check the usual properties of this protocol; in particular since t is chosen so that it is less than each prime factor in N, ability to answer more then one challenge unconditionally implies that the values hidden within Com and c are the same.