Secret-Key Zero-Knowlegde and Non-interactive Verifiable Exponentiation

Ronald Cramer and Ivan Damgård

BRICS[⋆], Aarhus University, Denmark
{cramer,ivan}@brics.dk

Abstract. We consider a new model for non-interactive zero-knowledge where security is not based on a common reference string, but where prover and verifier are assumed to possess appropriately correlated secret keys. We present efficient proofs for equality of discrete logarithms in this model with unconditional soundness and zero-knowledge. This has immediate applications to non-interactive verification of undeniable signatures and pseudorandom function values. Another application is the following: a set of l servers, of which less than $l/2$ are corrupt, hold shares of a secret integer s. A client C specifies g in some finite group G, and the servers want to allow the client to compute g^s non-interactively, i.e., by sending information to C only once. This has immediate applications in threshold cryptography. Using our proof system, the problem can be solved as efficiently as the fastest previous solutions that either required interaction or had to rely on the random oracle model for a proof of security. The price we pay is the need to establish the secret key material once and for all. We present an alternative solution to the problem that is also non-interactive and where clients need no secret keys. This comes at the expense of more communication and the assumption that less than $l/3$ of the servers are corrupt.

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. Many variants of the model have been studied, in particular variants where some extra resource is assumed to be available. In some cases, this allows to construct zero-knowledge proofs more efficiently than in the standard model, e.g., in terms of round or communication complexity. For instance, in the well known model of non-interactive zero-knowledge, prover and verifier are assumed to have access to a common random reference string σ. This allows the

[⋆] Basic Research in Computer Science (www.brics.dk), funded by the Danish National Research Foundation. Also supported by FICS, Foundations in Cryptography and Security, funded by the Danish Natural Sciences Research Council prover to prove his statement $x \in L$ simply by computing a single string π and send it to the verifier, who can check it against σ.

In this paper, we propose a new non-interactive variant, where there is no common random string (we do include a public string in our model for convenience, but this is not essential). The new ingredient is that prover and verifier are assumed to have secret keys sk_P, respectively sk_V. These are assumed to be chosen with some appropriate joint distribution depending on the language in question. The prover proves that $x \in L$ by computing a proof π from x, sk_P and some private information related to x. When π is sent to the verifier, he can check it against x and sk_V.

Intuitively, the prover is prevented from cheating because he doesn't know sk_V, and so does not know "how" the verifier will check the proof. On the other hand, although sk_P and sk_V must be correlated in a particular way, sk_V taken by itself has a distribution that is easy to simulate from scratch. Furthermore, we arrange it such that given sk_V and $x \in L$, the proof π that the prover would give is easy to compute, thus allowing the verifier's entire view to be simulated efficiently.

Our motivation for introducing this model is an efficient example we present allowing non-interactive proofs of statements related to discrete logarithms. We give here an informal presentation of the idea, which will be formalized later in the paper.

Let us assume that we have given a finite group G of prime order q, and that P has a secret number $s \in Z_q$. Now, sk_P is a random element $y \in Z_q$, while sk_V is a pair α, β where α is random in Z_q while $\beta = \alpha s + y$. We discuss later how such keys can be set up. Note that sk_V is independent of s. The purpose of the proof system is to allow P to prove that $g, h \in G$ satisfy $g^s = h$, whenever this is the case. To understand how sk_P, sk_V help to do this, think of s as a message, y as an authentication code, and α, β as a verification key. Indeed, if P were to reveal s, y, then V could check that s was in fact the value fixed at key set-up time by verifying $\beta = \alpha s + y$. It is easy to see that to cheat, the prover would have to guess α. Now, since the verification is done by taking a *linear* combination of s, y we can instead do the check "in the exponent" when we are given g^s instead of s. So given $g, h = g^s$, P sends as proof $\pi = g^y$, and V checks that $g^\beta = h^\alpha \pi$. Informally, this is zero-knowledge, since given g, h, sk_V, V can easily compute what π should be.

So the proof consists of sending one group element and requires one exponentiation to compute and at most two for verification. Later, we generalize the idea to arbitrary finite groups, where P, V do not even need to know the order of G. An important fact from a practical point of view is that neither the prover nor the verifier need random coins: security of the proof system relies only on the randomness involved in choosing the keys. Since obtaining random bits securely "on the fly" can be difficult, it is interesting to be able to push the need for randomness to a set-up phase.

We mention that the hash proof systems of Cramer and Shoup [1] are also a special case of our model where sk_P is empty. The most well-known example of hash proof systems also relate to equality of discrete logarithms: given generators g_0, g_1 of prime order group G, the prover can show for given h_0, h_1 that $h_0 = g_0^s, h_1 = g_1^s$. Here, sk_V consists of two random integers a, b, and the prover must know $g_0^a g_1^b$ to compute the proof. Thus, hash proof systems allow proving equality of discrete logs, assuming that the "base elements" g_0, g_1 are fixed.

Our proof system instead fixes the exponent, and allows the base to vary without changing the keys. This dramatically expands the range of possible applications, as we shall see. We emphasize that all our applications must of course assume that correctly chosen secret keys are set up for all would-be provers and verifiers before use. This can always be established by trusted parties, or by secure two-party or multiparty computation. For our main example proof system, we give an efficient key set-up protocol later in the paper. This protocol only involves the prover and verifier, it is constant round and has communication complexity $O(k)$ bits, where k is the security parameter.

An obvious application is to do non-interactive confirmation of undeniable signatures, when using Chaum's original scheme [2], or the convertible scheme of Rabin et al[9]. This is immediate because these schemes produce signatures by computing a group element from the input message and raising this to a fixed secret exponent. A further application is to verify outputs from the pseudo-random functions of Naor and Reingold[11]. A secret key for their construction consists of a set of fixed exponents, and one evaluates the function by raising a fixed element in a prime order group to a sequence of exponenents determined by the input. Using Nielsen's variant of this construction[12], it is safe to reveal the intermediate results. Each of these can be sent along with the function value and verified non-interactively using our proof system. This gives a functionality similar to verifiable pseudorandom functions, but the construction is conceptually simpler and more efficient than known constructions.

A final application is the following: a set of l servers, of which less than $l/2$ are corrupt, hold shares of a secret integer d. A client C specifies g in some finite Abelian group G, and the servers want to allow the client to compute g^d non-interactively, i.e., by sending information to C only once and with no communication between servers. This has immediate connections to threshold cryptography, and can be applied directly to distributed El-Gamal and RSA. Using our proof system, the problem can be solved as efficiently as the fastest previous solutions that either required interaction or had to rely on the random oracle model for a proof of security. The price we pay is, as mentioned, the need to establish the secret key material once and for all. Some variants are possible, however: a client without secret keys can still use the system, at the expense of an extra round of communication. Or he can do a key set-up protocol once and for all with the servers, and then use the system non-interactively.

A different type of "trade-off" is also possible. We present an alternative solution to the problem, which is not directly based on secret-key zero-knowledge, but uses a related technique. It is also non-interactive and needs no secret keys for clients. This comes at the expense of more communication and the assumption that less than $l/3$ of the servers are corrupt.

2 Secret-Key Zero-Knowlegde

Our model involves the following ingredients: interactive Turing Machines $\mathcal{P}, \mathcal{V}$ (Prover and Verifier) and the *Key generator* a PPT algorithm $\mathcal{G}$. In addition a (possibly infinite) set of strings PK.

In the model, we play initially a game where the language in which membership is to be proved is fixed, and where keys are set up: $\mathcal{P}, \mathcal{V}$ get as input $pk \in PK$ and 1^k where k is a security parameter. Then $\mathcal{P}$ outputs strings s, inp_P and $\mathcal{V}$ outputs string inp_v. Then $\mathcal{G}$ is run on input $1^k, pk, s, inp_P, inp_V$, and will output two strings sk_P, sk_V which will later be given to $\mathcal{P}, \mathcal{V}$, respectively.

The meaning of this is as follows: each pair s, pk, where $pk \in PK$, defines a language $L_{s,pk}$, that is, we assume there is a polynomial time algorithm that decides if $x \in L_{s,pk}$, when given s, pk as additional input. One can think of pk as a public key chosen once and for all, and s as a secret piece of information that the prover is committed to after the key set-up phase. Our model captures this by having P give s to G initially. Because the prover is committed to s, the language $L_{s,pk}$ is well defined, even though V gets no information initially on s. For instance, pk might specify a finite group, and s could be a secret discrete logarithm.

$\mathcal{G}$ models a protocol or a trusted party that will set up secret keys for $\mathcal{P}, \mathcal{V}$ which will help $\mathcal{P}$ in convincing $\mathcal{V}$ about membership in $L_{s,pk}$. The strings inp_P, inp_V allow us to model the influence that $\mathcal{P}$ or $\mathcal{V}$ are allowed on the keys produced.

Now, from inputs s, sk_P, pk and $x \in L_{s,pk}$ the prover computes output string $\mathcal{P}(x, s, pk, sk_P)$. This can be thought of as a non-interactive zero-knowledge proof that $x \in L_{s,pk}$. The verifier can from input x, a string pr (supposedly coming from $\mathcal{P}$) and pk, sk_V compute as output 1 for "accept" or 0 for "reject". We now have

Definition 1. *The triple* $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ *is said to be a* secret-key zero-knowledge proof system *for* PK *with error probability* $\epsilon(\cdot, \cdot)$ *if the following conditions are satisfied:*

Completeness. *Correct proofs produced and checked using matching keys are always accepted: Fix any* $pk \in PK$*, and any* s, sk_P, sk_V *that can be produced by honest* $\mathcal{G}, \mathcal{P}, \mathcal{V}$ *on input* pk*. Then for any* $x \in L_{s,pk}$*, we have* $\mathcal{V}(x, \mathcal{P}(x, s, pk, sk_P), pk, sk_V) = 1$ *with probability* 1.

Soundness. *Even given the secret prover information, no prover can produce* t *statements and proofs, and have any false statement accepted with probability better than* $\epsilon(pk, t)$*: Fix any* $pk \in PK$*, and for any (possibly unbounded) prover* P^**, set* $(s, inp_P) = P^*(pk)$*. Set* $(sk_P, sk_V) = \mathcal{G}(1^k, s, pk, inp_P, \perp)$*. Give* sk_P *as input to* P^**. Now do the following for* $i = 1...t$*:* P^* *produces a word* x_i *and a proof* pr_i*, and recieves* $\mathcal{V}$*'s output bit* $\mathcal{V}(x_i, pr_i, pk, sk_V)$*. We require that* $\mathcal{V}$ *rejects all* $x_i \notin L_{s,pk}$ *except with probability* $\epsilon(pk, t)$*.*

Zero-Knowledge. *The verifier's view of the key generation and proof of any true statement(s) can be simulated with the correct distribution. Fix any pair* (s, pk) $(pk \in PK)$*, and consider any verifier* V^**. Set* $inp_V = V^*(pk)$*,* *and* $(sk_P, sk_V) = \mathcal{G}(1^k, s, pk, \perp, inp_V)$. *Finally, for any word* $x \in L_{s,pk}$, *run* $\mathcal{P}(x, s, pk, sk_P)$ *to obtain a proof* pr. *There exists a PPT simulator* $\mathcal{M}_1$ *such that the output distribution* $\mathcal{M}_1(1^k, pk, inp_V)$ *is statistically indistinguishable from that of* sk_V. *Moreover, there exists a PPT simulator* $\mathcal{M}_2$ *such that the output distribution* $\mathcal{M}_2(1^k, pk, sk_V, x)$ *is statistically indistinguishable from that of* pr.

Discussion: In this model, the quality of the simulation is guaranteed by increasing the security parameter k. We do not require that the soundness error vanishes with increasing k, but this can be achieved by generating several independent sets of keys for the same pair s, pk and repeating the proof system in parallel. However, for all the applications we are aware of, this is not necessary, because the application allows us to choose s, pk such that the soundness error is already exponentially small for all polynomial t.

The simulator is given the public string pk and must simulate w.r.t. pk. Thus, unlike standard non-interactive zero-knowledge, the reason why the simulator can work efficiently is not that it gets to choose the public string by itself, but that it knows sk_V and can use this knowledge when simulating the proofs.

Note that the zero-knowledge requirement implies that a cheating verifier's view can be simulated, even in a case where several statements are proved after key generation and where the verifier can decide the order in which they are proved: one simply runs $\mathcal{M}_1$ and then $\mathcal{M}_2$ a number of times using the output of $\mathcal{M}_1$ and the relevant statements to be proved. This works since the honest prover acts independently on each statement, given his secret key.

The definition requires unconditional security for both parties. This is possible since both players possess information that is information theoretically hidden from the other player. However, if $\mathcal{G}$ is realized via a protocol that only offers computational security, this will of course reduce the security of the proof system to computational as well.

Our model may superficially resemble earlier proposals for "zero-knowledge with preprocessing". The essential difference is that we have no restriction on the number of proofs that can be done based on a given key pair, while earlier schemes used the preprocessing phase to build resources that would eventually run out later.

We proceed to present our main example of SKZK. Our set of public keys PK is the set of strings pk that contain (in some fixed format) a specification of a finite Abelian group G and natural numbers k_0, k_1, where the smallest prime factor in the order of G is larger than 2^{k_0}. The language $L_{s,pk}$ consists of pairs of elements $g, h \in G$ such that $h = g^s$, provided that s is an integer in $[0..2^{k_1} - 1]$. Otherwise it is empty.

This specification reflects the fact that a bound on the size of prime factors in the order of G will be needed to estimate the soundness error of our proof system, and that it is only intended to work for values of s up to a certain limit.

The specification of G is a string, such that if it is known, one can decide membership in G and compute the group operation and inverses in G efficiently (poly-time in the size of the specification). For instance, G could be a prime order subgroup of Z_p^* for some prime p, or (a subgroup of) Z_n^* for an appropriately chosen RSA-modulus n.

The key generator is given s, G, k_0, k_1, security parameter k, and two strings inp_P,inp_V that are interpreted as integers in the standard way (recall that these are used to model the allowed influence of (corrupt) P or V on the choice of keys). Test if the following conditions are satisfied: $s \in [0..2^{k_1} - 1]$, $inp_P \in [0..2^{k_0+k_1+k}]$ or inp_P is empty, $inp_V \in]0..2^{k_0}]$ or inp_V is empty.
If the conditions are satisfied, then set $\alpha = inp_V$, or if inp_V is empty, choose α uniformly random in $]0..2^{k_0}]$. Set $y = inp_P$, or if inp_P is empty, choose y uniformly random in $[0..2^{k_0+k_1+k}]$. Set $\beta = \alpha s + y$. Finally, set $sk_P = (s, y)$, $sk_V = (\alpha, \beta)$ and output these values.
If the conditions on s, inp_P, inp_V are violated, output empty strings and stop.

The honest prover and verifier are assumed to always choose empty strings as inp_P, inp_V.

From a practical point of view, this SKZK proof system can be used to allow a prover to get $g \in G$ as input, send g^s to the verifier and non-interactively prove that this was correctly done. The specification of $\mathcal{G}$ allows that a corrupt P can choose s, y (in the correct intervals), but will get no information on α. A corrupt V can choose α as any value in the interval he likes, but he learns no information on s, y other than β. The specification also allows a corrupt party to block the key generation, this models the fact that since we want to implement $\mathcal{G}$ via a two-party protocol, we cannot guarantee successful termination because one player can just stop early.

We proceed to describe the other algorithms:

The prover will on input g, h where $h^s = g$, compute $v = g^y$ as the proof (assuming sk_P is not empty).
The verifier will on input g, h, v check whether $g, h \in G$ and $g^\beta = h^\alpha \cdot v$, and will accept if and only if this is the case.

We have:

Theorem 1. *The above is a SKZK proof system for $\mathcal{L}$ with error probability $t/(2^{k_0} - t)$.*

Proof. Completeness is trivial by simply plugging the values produced by the prover into the equation checked by the verifier. For soundness, assume first that $t = 1$, that G is cyclic, and that we have $h \neq g^s$ and some proof v. Writing everything as a power of a generator a of G, we have $g = a^i, h = a^j, v = a^m$. The assumption $h \neq g^s$ implies $si - j \neq 0 \bmod q^l$ for some prime factor q in the order of G, where q^l is the maximal q-power dividing $|G|$. In order to have the proof accepted, the prover must arrange it such that

$$g^\beta = h^\alpha v$$

which means that $\beta i = \alpha j + m \bmod q^l$. Now, since key generation ensures that $\beta = \alpha s + y$ we find that

$$\alpha(si - j) = (m - iy) \bmod q^l.$$

Let q^b be the maximal q-power dividing both sides of this equation. By choice of α, it is non-zero modulo q, so we have

$$\alpha \frac{si - j}{q^b} = \frac{m - iy}{q^b} \bmod q^{l-b}.$$

The assumption $si - j \neq 0 \bmod q^l$ implies $b < l$. It follows that

$$\alpha = \frac{m - iy}{q^b} \cdot (\frac{si - j}{q^b})^{-1} \bmod q^{l-b}.$$

in other words, to have the false g, h accepted, the prover must guess $\alpha \bmod q^{l-b}$. However, α was randomly chosen among $2^{k_0} < q$ possibilities, and by the specification of G, the prover has no a priori information on α. So accept happens with probability at most 2^{-k_0}. If G is not cyclic, we can write G as a direct product of r cyclic components $G_1, ..., G_r$ and g, h as r-tuples $(g_1, ..., g_r), (h_1, ..., h_r)$ in the standard way. If $h \neq g^s$, this means that $h_i \neq g_i^s$ for some i, and we can then use the argument above in the cyclic subgroup G_i.

Finally we consider the case of proving several statements: if a cheating prover sends any correct g, h where $h = g^s$, he can compute from his secret key what the correct proof should be, and since this is the only value the verifier will accept, the prover can predict the verifier's reaction to any proof he might choose to send along with g, h.

Now consider the situation where the prover is about to compute a new proof, assuming that he has not yet made the verifier accept a false statement. By the above, the new information the prover could have learned earlier must come cases where a proof of a false statement was rejected. Assuming that t false proofs were already rejected, the prover can exclude t possible values of α, so the next proof will be accepted with probability at most $1/(2^{k_0} - t)$. This implies the claimed error probability by an easy induction argument.

Zero-knowledge follows since we can simulate the choice of α, β by first choosing $\alpha \in]0..2^{k_0}]$ based on inp_V in the same way as $\mathcal{G}$ would have done it. Then we choose at random $\beta \in [0..2^{k_0+k_1+k}]$. This simulates α perfectly and β with statistically close (in k) distribution, since in real life $\beta = \alpha s + y$ and y is k bits longer than αs. Furthermore, given correctly distributed α, β and $(g, h) \in L$, the (uniquely determined) proof that the honest prover would send is $v = g^\beta h^{-\alpha}$.

2.1 Some Variations

From the proof of the above theorem, it is clear that we do not really need to fix the group G in advance. The same key set-up can be reused for any Abelian group, the only price to pay may be that the soundness error probability can be larger: if the group has a prime factor q in its order smaller than 2^{k_0}, the error probability for one proof will be $\theta(1/q)$.

A variation on this: Suppose G is a direct product $G = H \times K$, where $|H|$ has only primes factors $> 2^{k_0}$. And furthermore for some publically known γ, it holds that $e^\gamma = 1$ for all $e \in K$. Then given an instance (g, h), we can use the original proof system on the pair (g^γ, h^γ), in order to prove that $g^{s\gamma} = h^\gamma$. For some applications, including threshold RSA, this is sufficient.

Finally, we note that some generalizations are possible of the form of statement proved: suppose we have two secrets s, s' and have set up keys y, y' and $(\alpha, \beta), (\alpha', \beta')$ just as above, except that we have designed the key generation such that $\alpha = \alpha'$. It is then possible for the prover to send g, g', h and prove that $h = g^s g'^{s'}$. The proof would be $v = g^y g'^{y'}$ and the verifier would check that $g^\beta g'^{\beta'} = v h^\alpha$.

3 Key Set-Up Protocol

Suppose now that P, V want to agree on a set of keys for the SKZK proof system we have described, assuming that the public string pk has already been generated (i.e., some group has been chosen) and P knows the secret s he will be using. We sketch here an efficient protocol that that securely realizes the $\mathcal{G}$ we specified earlier.

The protocol can be proved secure in Canetti's model for secure function evaluation[14], assuming a static adversary that corrupts P or V. We make no claims here on composability of the protocol, other than the sequential composability that follows from Canetti's definiton. However, we believe that in the common reference string model and using the techniques from [7], a universally composable version could be designed without essential loss of efficiency.

3.1 A First Attempt

We first describe a solution that works if both parties follow the protocol. Suppose V chooses a key pair for a semantically secure and additively homomorphic public-key cryptosystem. As example we will use the one by Paillier[13]. He sends the public key pk_V to P, and also sends the encryption $E_{pk_V}(\alpha)$ where α has been chosen as described in the key generation for the SKZK proof system.

Then (assuming P knows s already) P chooses y as in the key generation for SKZK, uses the homomorphic property to compute an encryption $E_{pk_V}(s\alpha + y)$ and sends this to V. Finally, V decrypts and defines the result to be β. Of course, we want that $\beta = s\alpha + y$ as integers, and the Paillier cryptosystem is only homomorphic w.r.t. addition modulo some RSA modulus - but as long as the modulus is chosen large enough compared to the sizes of α, s and y, no modular reductions will occur, and β will be the correct value.

Clearly, V learns nothing new except β, and a computationally bounded P learns nothing new, assuming he cannot break the semantic security.

3.2 The Real Solution

In order to make a solution secure even against active cheating, we assume that we have available a public key pk_C for an integer commitment scheme such as the one by Damgård and Fujisaki[3], allowing P or V to commit to an integer a of any size and prove efficiently in zero-knowledge that a belongs to some interval using the technique of Baudot[4]. We discuss below where pk_C could come from.

Note that this commitment scheme is homomorphic: from commitments that can be opened to integers a, b it is easy to compute a commitment that can be opened to (only) $a + b$. It is also trapdoor, i.e., knowing a certain piece of side information, it is possible to produce a commitment that can be opened to any desired value. Notation: $Com_{pk_C}(x, r)$ denotes a commitment under public key pk_C to x using random coins r.

A final tool we need is the efficient method outlined in [6] allowing a party to make public a Paillier encryption $E_{pk_V}(\alpha)$ and prove that α belongs to a given interval. This involves making a commitment $Com_{pk_C}(\alpha, r_\alpha)$, proving that it contains the same value as $E_{pk_V}(\alpha)$ and proving that α is in the correct interval using the technique from [4]. For details see [6].

Then we do the following:

1. V sends the key pk_V, the encryption $E_{pk_V}(\alpha)$ and proves in ZK that α is in the correct interval.
2. P chooses s, y as in the key generation for SKZK, makes commitments $S = Com_{pk_C}(s, r_s), Y = Com_{pk_C}(y, r_y)$ and proves that he knows how to open these commitments to integers in the correct intervals. Similarly, he chooses $\bar{s}, \bar{y}$ as random numbers $2k$ bits longer than s respectively y, makes commitments $\bar{S} = Com_{pk_C}(\bar{s}, r_{\bar{s}}), \bar{Y} = Com_{pk_C}(\bar{y}, r_{\bar{y}})$, and proves that $\bar{s}, \bar{y}$ were chosen in the correct intervals.
3. P uses the homomorphic property of the encryption scheme to compute encryptions $E_{pk}(\alpha s + y), E_{pk}(\alpha\bar{s} + \bar{y})$, and sends these to V, who decrypts to get results β, respectively $\bar{\beta}$.
4. V sends a random k-bit challenge e. Both parties use the homomorphic properties of the commitment scheme to compute from $S, Y, \bar{S}, \bar{Y}$ commitments $Z_s, Z_y$to $z_s = \bar{s} + es, z_y = \bar{y} + ey$. P opens Z_s, Z_y to reveal z_s, z_y to V.
5. V checks that the openings were correct, and that $\bar{\beta} + e\beta = \alpha z_s + z_y$. If so, he accepts using α, β as keys to check proofs from P in the future. Output for P is s, y.

Given an oracle that supplies pk_C, this protocol can be proved to securely realize $\mathcal{G}$ as specified above in Canetti's model for secure function evaluation[14], assuming a static adversary that corrupts P or V. Due to space limitaitons, we only give informally the essential ideas needed for this:

By inspection, it is trivial to check that V always accepts if both parties follow the protocol, and that the outputs generated have the same distribution as $\mathcal{G}$ would have produced.

If a party is corrupt, we need to describe a simulator that interacts one one side with the corrupt player and on the other side with the "ideal function" $\mathcal{G}$

as specified above. It must create a view for the corrupt player that is indistinguishable from the real conversation, and at the same time interact with $\mathcal{G}$ on behalf of the corrupted player. The induced input/output behavior of $\mathcal{G}$ must be consistent with the view generated for the corrupted player. In general, if this game comes to a point where the corrupt player would make the honest player reject and stop, the simulator handles this by sending an illegal value as input to $\mathcal{G}$. This causes $\mathcal{G}$ to stop without generating output, which is consistent with what happens in real life.

Now, assume P is honest and V may actively cheat. The simulator can use rewinding to extract α from the ZK proof of knowledge given in Step 1 and give this to $\mathcal{G}$ as inp_V. Note that it happens with only negligible probability that α is an illegal value simultaneously with the proof being accepted. So we may assume that α is in the correct interval, and and $\mathcal{G}$ will return β to the simulator. From the protocol description, it then follows that $\beta, \bar{\beta}$ have distribution statistically close to that of $y, \bar{y}$, i.e., uniform and independent. In particular, they convey only negligible information on $\bar{s}$. It follows that z_s has distribution statistically close to that of $\bar{s}$, i.e., uniform and independent from $y, \bar{y}$. Finally, z_y always satisfies $\bar{\beta} + e\beta = \alpha z_s + z_y$. It follows that the opened values V sees in the protocol can be simulated with statistically close distribution by choosing $\bar{\beta}, z_s$ uniformly and independently with the same distribution as $y, \bar{y}, \bar{s}$ and setting $z_y = \bar{\beta} + e\beta - \alpha z_s$. So if the simulator knows the trapdoor for pk_C, it can simulate efficiently V's view of the protocol given only pk, pk_C.

Then, assume that V is honest. P's view of Step 1 can be simulated by supplying a random encryption and commitment and simulating the zero-knowledge proof to be given. Step 2 forces P to choose values $s, y, \bar{s}, \bar{y}$ in the correct intervals, and these values can be extracted by a simulator using the ZK proofs of knowledge given in Step 2, and s, y can be given as input to $\mathcal{G}$. Note that in the following steps, P learns no new information, the simulator can just play the game following V's part of the protocol. Hence, the only remaining question is whether the protocol ensures that $\beta = \alpha s + y$. The probability that the protocol completes successfully while this condition is violated must be negligible since otherwise it will not be consistent with what the ideal $\mathcal{G}$ produces. We argue that if $\beta \neq \alpha s + y$, then V accepts with negligible probability. For this, it is sufficient to show that if in Step 4, P can give satisfying answers to a non-negligible fraction of the possible challenges, then $\beta = \alpha s + y$. Under this assumption, by rewinding P, we can efficiently obtain acceptable replies to two distinct values e, e'. Because V accepts in both cases, P has opened values z_s, z_y, z'_s, z'_y such that

$$\bar{\beta} + e\beta = \alpha z_s + z_y \quad \bar{\beta} + e'\beta = \alpha z'_s + z'_y$$

from which we conclude that

$$(e - e')\beta = \alpha(z_s - z'_s) + z_y - z'_y$$

Now, by the binding property of the commitment scheme, except with negligible probability, it holds that $z_s = \bar{s} + es, z_y = \bar{y} + ey, z'_s = \bar{s} + e's, z'_y = \bar{y} + e'y$. Plugging this in, we immediately obtain $\beta = \alpha s + y$ as desired.

On efficiency, it is straightforward to check by inspection of the above and [4, 6,3] that the protocol requires communicating only a constant number of encryptions and commitments, and can be executed in a constant number of rounds.

Finally, we discuss how to set up the key pk_C. This key consists of an RSA modulus n and two elements $g_0, h_0 \in Z_n^*$ with only "large" prime factors in their order, and such that h_0 is in the group generated by g_0. Fortunately, in our main application, namely threshold RSA, an RSA modulus n is already available. Therefore the key set-up will work, assuming that elements g_0, h_0 have been chosen once and for all. It requires only little effort to do this at the time when n is set up. For instance, if n is a product of safe primes, simply choosing g_0, h_0 as random squares will be correct, except with negligible probability.

Another possibility consists in letting P choose a public key w.r.t. which V can commit, and vice versa. Two-party protocols for setting up a key in this way are described in detail in [3]. Compared to the previous solution, this costs a factor of k in round- and communication complexity, but does not assume any previous key set-up at all.

4 Applications

4.1 Undeniable Signatures

In the original scheme for undeniable signatures by Chaum [2], the public key is a safe prime p, i.e., such that $(p-1)/2 = q$ is also a prime, and elements $g, h \in Z_p^*$, where s, such that $h = g^s \bmod p$ is the private key. A signature on message m is $h(m)^s$, where h is some appropriate hash function that maps into Z_p^*. Signatures seem to hard to forge under the Diffie-Hellman assumption, but furthermore the idea is that it is hard to verify a signature unless the signer is willing to help you, by engaging in a protocol where he proves that the discrete log of h base g equals that of z base $h(m)$ where z is the purported signature. This is called a confirmation protocol.

Clearly, our proof system can be directly used to build a non-interactive confirmation protocol for this scheme, which was not known before, except in the random oracle model. Furthermore, it also applies to the convertible scheme of Rabin et al. [9], since this scheme is essentially the same but where Z_p^* is replaced by Z_n^*, where n is a safe prime product. The idea being that by revealing the "public exponent" corresponding to s, all signatures can be instantly converted to ordinary signtures. Some minor technical problems, related to the fact that the order of Z_n^* contains a small prime factor 2, are handled in [9], and their solutions to this translate easily to our case.

4.2 Pseudorandom Functions

In [11], Naor and Reingold present a pseudorandom function construction based on the DDH assumption. The construction takes place in the same group Z_p^* mentioned above. This has proved a very useful idea for making efficient protocols, for instance, Nielsen [12] describes a variant that can also be computed in a threshold fashion, and shows how this can be used to build efficient asynchronous Byzantine agreement protocols and threshold RSA signatures without random oracles.

The variant from [12] has a private key k consisting of l pairs of random elements from Z_q, $(\alpha_{1,0}, \alpha_{1,1}), ..., (\alpha_{l,0}, \alpha_{l,1})$. Also, a random public $g \in Z_p^*$ of order q is given. The function can take any string $\sigma = (\sigma_1, .., \sigma_m)$ where $m \leq l$ as input, and the output is

$$f_k(\sigma) = g^{\prod_{i=1}^m \alpha_{i,\sigma_i}}$$

Clearly, our proof system can be used to set up key pairs allowing the party who knows the private key k to prove that some element in the subgroup of order q has been raised to powers $\alpha_{j,b}, j = 1..l, b = 0, 1$, respectively.

This leads to a way to non-interactively verify values of f_k when evaluated on strings of length precisely l. Namely, on input $\sigma = (\sigma_1, ..., \sigma_l)$, send

$$g^{\alpha_{1,\sigma_1}}, g^{\alpha_{1,\sigma_1}\alpha_{2,\sigma_2}}, ..., g^{\prod_{i=1}^m \alpha_{i,\sigma_i}} = f_k(\sigma)$$

plus a proof that the j element on the list is the previous one raised to α_{j,σ_j}. Note that the first elements on the list are f_k evaluated on substrings of σ, so it is secure to reveal these by pseudorandomness of $f_k()$. Some applications allow to evaluate the function on consecutive values $0, 1, 00, 01, 10, 11, 000, ..$, or in general such that we never evaluate the function on an input that is a prefix on a previously calculated value. In this case, is secure to use the domain of all strings of length at most l. With consequtive values, one can exploit the fact that most of the required list of function values needed to verify a new one are already known, so only a single new value and proof needs to be sent.

This gives a functionality similar to that of verifiable pseudorandom functions (VRF), as proposed by Micali, Rabin and Vadhan[10], although of course at the expense of having to set up keys for our proof system first. With a VRF, one can simply publish a public key and then send function values and non-interactive proofs of correctness. However, VRF's are only known to exist under the strong RSA assumption, or under various strong and non-standard variants of the DH/DDH assumptions [5,8]. Moreover, most of these solutions are rather complicated and inefficient – with the exception of [8]. An alternative to the VRF concept would be to commit on the key and use standard non-interactive zero-knowledge to prove that the funcion value is correct, but this would be very inefficient. In contrast, our technique allows us to assume only standard DDH and have a reasonably efficient and conceptually simple solution.

It is easy to adapt our technique also to the threshold pseudorandom function from Nielsen[12]. This gives a non-interactive solution with a smaller communication complexity than the interactive protocol from [12].

5 Non-interactive Verifiable Exponentiation

We consider the following problem: a set of l servers, of which t are corrupt, hold shares of a secret integer d. A client C specifies g in some finite Abelian group

G, and the servers want to allow the client to compute g^d non-interactively, i.e., by sending information to C only once and with no communication between servers. This has immediate connections to threshold cryptography, and can be applied directly to distributed El-Gamal and RSA. Below, we two solutions with different properties.

5.1 Using Secret-Key Zero-Knowledge

To illustrate how we can use secret-key zero-knowledge in this context, the easiest way is to consider Shoups threshold RSA scheme[15], where indeed the purpose is to do non-interactive verifiable exponentiation in the group Z_n^*, where $n = pq$ is a product of safe primes, and where we assume that $t < l/2$. To make this scheme robust (verifiable), each server S_i needs to prove that a given input number was raised to a secret share s_i (of the private RSA exponent) held by S_i. By squaring the inputs, Shoup makes sure that this proof can be done assuming we work in a group with only large prime factors in its order. It is therefore clear that our proof system can be directly plugged in, instead of the random oracle based proofs that were used in [15]. This will even be more efficient by a constant factor.

Of course, this can only used directly assuming there are keys set up for proofs going from each server to the client. But we can also do something assuming the client has no keys, but we have keys for pairwise interaction between the servers. Namely, the clients requests from each server a signature share ($g^{s_i} \bmod n$) and proofs of correctness for this share, directed to each of the other servers. Then the client sends these signature shares and proofs back to the severs for approval. He will only keep those signature shares that were approved by a majority of the servers. By soundness of the proofs, this will leave the client with at least $t + 1$ shares, all of which are correct, and this is sufficient to find $g^d \bmod n$.

5.2 An Alternative without Secret Key Zero-Knowledge

The following solution is non-interactive and does not require the client to have any secret keys. This comes at the price of more communication and assuming $t < l/3$. For simplicity, we work over a group G_q of prime order q, and the secret value d is an element of Z_q.

Consider first a situation where some server S knows a secret value $\hat{d}$, and where the other $l-1$ servers S_r have correct shares $\hat{s}_r$ in $\hat{d}$, according to Shamir's scheme. S also knows the polynomial $F(X) = \hat{d} + d_1 X + \cdots + d_t X^t$ according to which $\hat{d}$ was shared. Write $\hat{s}_r = F(r)$.

Here is a simple protocol where the client C can easily check whether the value h he receives from S is indeed equal to $g^{\hat{d}}$, with $g \in G_q$ specified by the client. There is no interaction between the servers. Each of the servers just sends some information to C, and C performs an off-line check on the total of this information to decide on the correctness of h. The protocol has zero error probability, while C learns only the value $g^{\hat{d}}$.

S sends the value h to C, equal to $g^{\hat{d}}$ if S is honest. Additionally, S sends the values h_j, equal to g^{d_j} if S is honest. Each other server S_r sends the value f_r to C, equal to $g^{\hat{s}_r}$ if S_r is honest.

From the information sent by S and by performing "polynomial evaluation in the exponent," C now computes the values f'_r, equal to $g^{\hat{s}_r}$ if S is honest. Concretely, C computes

$$f'_r = h \cdot \prod h_j^{r^j}. \tag{1}$$

In the case that there are at most t inconsistencies

$$f_r \neq f'_r,$$

C decides that $h = g^{\hat{d}}$ indeed. Otherwise he decides that S is corrupt.

It is easy to see that this works. First, consider the case that S is honest. This implies that h and the h_j are correct. If S_r is honest as well, then clearly $f_r = f'_r$. Up to t of the servers S_r are corrupt, so there are at most t inconsistencies $f_r \neq f'_r$. Thus, C makes the correct decision.

Second, if S is corrupt and $h \neq g^{\hat{d}}$, then there are more than t inconsistencies and C correctly decides that S is corrupt. This is argued as follows. The information sent by S does define, in "the exponents," a polynomial of degree at most t. However, since $\log_g h \neq \hat{d}$ by assumption, it must be a different one from $F(X)$. By Lagrange interpolation and the natural one-to-one correspondence between Z_q and G_q, it follows that at most t of the $l-1$ values f'_r equal $g^{\hat{s}_r}$. Equivalently, $f'_r \neq g^{\hat{s}_r}$ for at least $(l-1-t)$ values of r. However, apart from S, there may be $t-1$ other corrupt servers S_r. Therefore, $f_r \neq f'_r$ for at least $(l-1-t)-(t-1) = l-2t$ values of r. But $t < l/3$, so this means that there are more than t inconsistencies, and that C decides that S is corrupt.

Finally, we argue that a static adversary who corrupts C and at most t servers (but not S) will not learn nothing except $g^{\hat{d}}$. We do this by simulating his entire view given this value. From corrupting t servers the adversary will learn $\hat{s}_r$, for t values of r. Suppose without loss of generality that these are $\hat{s}_1, ..., \hat{s}_t$. This can be simulated perfectly by choosing t uniformly random values modulo q. These values together with $\hat{d}$ define a polynomial $F()$ of degree $\leq t$ where $F(0) = \hat{d}, F(1) = \hat{s}_1, ..., F(t) = \hat{s}_t$. Since we have $t+1$ values of $F()$, it follows that for any coefficient d_j of $F()$, there exists Lagrange interpolation coefficients $\gamma_0, ..., \gamma_t$ such that

$$g^{d_j} = (g^{\hat{d}})^{\gamma_0} \prod_{i=1}^{t} (g^{\hat{s}_i})^{\gamma_i}$$

and this value can easily be computed given $g^{\hat{d}}, \hat{s}_1, ..., \hat{s}_t$, i.e., we can simulate perfectly the extra information sent by S. Finally, we can use these values to simulate the contribution from honest S_r's using (1).

Now we return to the scenario of interest, non-interactive verifiable exponentiation. Each of the l servers S_i has a share s_i of d, according to Shamir's scheme with $t < l/3$. Let $g \in G_q$ be the element specified by the client C.

Additionally we now assume that, for each server S_i, an instance of the above verification protocol has been correctly set up, where S_i plays the role of S, the other servers play the roles of the S_r, and $\hat{d}$ is replaced by s_i.

If we now run the verification protocol above for each server S_i, the client C can easily filter out an incorrect value sent by a corrupt S_i, and remain with at least $l - t > t$ correct values g^{s_i}. By "interpolation in the exponent," i.e., multiplying these correct values g^{s_i} together, raised to appropriate Lagrange interpolation coefficients, C recovers the correct value g^d.

References

1. Ronald Cramer, Victor Shoup: Universal Hash Proofs and a Paradigm for Adaptive Chosen Ciphertext Secure Public-Key Encryption. Proc. of EUROCRYPT 2002: 45–64, Springer Verlag LNCS.
2. David Chaum, Hans Van Antwerpen: Undeniable Signatures. Proc. of CRYPTO 1989, Springer Verlag LNCS. pp.212–217.
3. Ivan Damgård, Eiichiro Fujisaki: A Statistically-Hiding Integer Commitment Scheme Based on Groups with Hidden Order. Proc. of ASIACRYPT 2002: 125–142, Springer Verlag LNCS.
4. Fabrice Boudot: Efficient Proofs that a Committed Number Lies in an Interval. Proc. of EUROCRYPT 2000: 431–444, Springer Verlag LNCS.
5. Anna Lysyanskaya: Unique Signatures and Verifiable Random Functions from the DH-DDH Separation. Proc. of CRYPTO 2002: 597–612, Springer Verlag LNCS.
6. Ivan Damgård, Mads Jurik: Client/Server Tradeoffs for Online Elections. Proc. of Public Key Cryptography 2002: 125–140. Springer Verlag LNCS.
7. Ivan Damgård and Jesper Nielsen: *Efficient Universally Composable Multiparty Computation*, proc. of Crypto 03, Springer Verlag LNCS.
8. Yevgeniy Dodis: *Efficient Construction of (Distributed) Verifiable Random Functions*, Proc. of PKC 2002, Springer Verlag LNCS.
9. Rosario Gennaro, Tal Rabin, Hugo Krawczyk: RSA-Based Undeniable Signatures. Journal of Cryptology 13(4): 397–416 (2000).
10. Silvio Micali, Michael O. Rabin, Salil P. Vadhan: Verifiable Random Functions. Proc. of IEEE FOCS 1999: 120–130.
11. Moni Naor, Omer Reingold: Number-theoretic Constructions of Efficient Pseudo-random Functions. Proc. of IEEE FOCS 1997: 458–467.
12. Jesper Buus Nielsen: A Threshold Pseudorandom Function Construction and Its Applications. Proc. of CRYPTO 2002: 401–416. Springer Verlag LNCS.
13. Pascal Paillier: Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. Proc. of EUROCRYPT 1999: 223–238, Springer Verlag LNCS.
14. Ran Canetti: Security and Composition of Multiparty Cryptographic Protocols. Journal of Cryptology 13(1): 143–202 (2000).
15. Victor Shoup: Practical Threshold Signatures. Proc. of EUROCRYPT 2000: 207–220, Springer Verlag LNCS