Unclonable Group Identification

Ivan Damgård, Kasper Dupont, and Michael Østergaard Pedersen

Aarhus University, BRICS

Abstract. We introduce and motivate the concept of unclonable group identification, that provides maximal protection against sharing of identities while still protecting the anonymity of users. We prove that the notion can be realized from any one-way function and suggest a more efficient implementation based on specific assumptions.

1 Introduction

A large body of literature studies the problem of group identification, where one wants to verify that a given user is a member of a certain group, while ensuring that the user's personal identity is not revealed. Particular instances of this include group signatures [5, 3, 22] and identity escrow[16]. In some applications, a dishonest user has an interest in giving away to another person the data that allow him to identify himself as a member of the group - such as password and secret keys. The security problems implied by such a scenario have not been given much attention so far in the literature[1].

In this paper we study this type of problem. As a motivating example, consider the issue of software protection: it is well known that one of the strongest motivating factors in getting people to register as software users is if this enables some functionality that cannot be accessed without registration (and payment). This works particularly well, if the functionality requires access to the vendor's website, since then reverse engineering the software is not sufficient to get unauthorized access to the functionality. In the case of games, for instance, the opportunity to play against others may be available to only registered users, and only through the vendor's website.

Verifying that a user is registered may be done in many different ways. In this paper, we are interested in solutions that work under the following constraints:

– An honest user can connect an unlimited number of times using the same private key material, or at least updates should only be necessary with long time intervals.
– We want to protect users' privacy, i.e., honest users have to identify themselves *only* as registered users and do not have to reveal their personal identities.

[1] Some earlier works suggest to discourage this by forcing users to either give away *all* their information, or nothing, but here we are interested in cases where dishonest users in fact have an interest in giving everything away.

– We want to do as much as possible to protect against attacks where a user "clones" himself by handing a copy of his personal data (software, secret key(s), etc.) to another person in order to get the benefits of two registrations while only paying for one.

Note that the cloning attack may be easy or very hard to carry out physically, depending on how the user's personal keys are stored, but only in very few cases can it be considered impossible.

Of course, we can only hope to detect cloning if the user and clone actually connect to the vendor's website. A further trivial observation is that if first the user connects, then leaves the site and then the clone connects, we cannot distinguish this from two connections made by an honest user, since he would also use the same private key material in both cases. An event we *can* hope to detect, however, is if both user and clone connect so that they are on the site simultaneously, since this is exactly what cannot occur if the user has been honest. In this case, we not only want to detect the attack, we also want to be able to reveal the identity of the user who cloned himself. Note that, apart from the fact that the above simultaneous scenario is the only one in which we can hope to catch a cloning attack, the scenario is also of practical relevance. For instance, the case of a user who buys one copy of a game and distributes it to all his friends so they can play against each other online, is exactly a case where a number of clones would want to be connected simultaneously.

An *unclonable identification scheme* informally is an identification scheme where honest users can identify themselves anonymously as members of a group, but where clones of users can be detected and have their identities revealed if they identify themselves simultaneously. In this paper, we give a formal definition of this primitive. We show that it can be realized assuming existence of one-way functions (which is clearly a minimal assumption), and we give a more efficient implementation based on specific assumptions. On the technical side, our most efficient solution is based on a new technique for proving in zero-knowledge, given g^x in a group of prime order, that x was chosen pseudorandomly from on a committed secret key.

Of course, before attempting a construction such as we have sketched, one should verify if existing primitives already allow solving the problem. First, one might consider using an anonymous E-cash scheme[17, 6], i.e., some number of electronic coins are issued to each user, and users use them to "pay" for access to the site. This would lead to a functionality that is incomparable to the one we sketched above: Cloning in this case means sharing e-coins with others, and so the cloning attack is exactly double spending and can therefore detected even if the two spendings do not take place simultaneously. But on the other hand, honest users can only use each coin once, and must therefore either possess a very large secure memory, or come back for more coins throughout the life of the system. This reveals information on how often a user connects, and is also not consistent with our goal, namely a solution where you can join a group once and then identify yourself an unlimited number of times using the same key material.

One may also consider using group signatures[5, 3, 22], and have users identify themselves by signing a message chosen by the verifier (using his current system time, for instance). This achieves anonymity but does not protect against cloning. To do this, one would need the property that if the same user signs the same message twice, this would result in signatures that could be detected as coming from the same user. This does not follow from the standard definition of group signatures, and is actually false for known schemes, since these are probabilistic and produce randomly varying signatures even if the message is fixed. A similar comment applies to identity escrow schemes[16].

2 Definition

An unclonable identification scheme involves a *Group Manager* GM, a set of *Verifiers* and some number of *Users*. The idea is that after some initialization, there will be several events, where some set of users prove "at the same time" to a verifier V that they are members of the group managed by GM. Since we want to detect if V is talking to clones of the same user at the same time, every proof should take as input some string α that represents in some sense the current time or phase of the protocol we are in. However, this does not have to be linked to *real time*. What is important is that whenever a set of users want to prove themselves, they should agree with V on a value for α that has not been used before. More precisely, the demands are

- An honest V must be able to ensure that all users he talks to at a given point prove themselves using the same value of α.
- An honest user should be able to ensure that he never executes $Prove$ with the same value of α more than once.

One solution that works in the case where V runs a website that users would like to be connected to for some length of time, is as follows: with regular intervals, e.g., each hour each user who is connected must prove himself using the current date and hour as α, as defined by the verifier's system time. This works if there is sufficient agreement on the time between users and V and if users remember at which time they last did a proof. But many other solutions are possible. Therefore, we have chosen to separate the way time is defined from the definition as such by assuming that the entire system proceeds in consecutive *phases*, with a unique number assigned to each phase. In each phase, some subset of users decide to prove themselves to some verifier V, and the number assigned to the current phase will be used as the string α. In the full version of this paper, [10] we propose a way to realize such a scenario without relying on synchronization, or requiring users to keep state.

The system is defined by probabilistic polynomial time algorithms $KeyGen$, *Detect* and two-party protocols $Join$ and $Prove$. These are used as follows:

- Initially, GM runs $KeyGen$ on input 1^k, to get output public key pk and secret key sk. We assume for simplicity that the set of possible pk's output by $KeyGen(1^k)$ can be recognized in polynomial time.

- When a user U joins the system he runs *Join* with GM. Common input is *pk*. Private input to GM is *sk*. The protocol outputs to GM either "reject" or a string *id*. Output to U is "reject" or a membership certificate $cert_U$. We assume *Join* is executed on a secure channel so that no other entity will have access to the data exchanged.
- To prove he is a member of the group, the user U executes protocol *Prove* with a verifier V. Common input is the public key *pk* and the string α assigned to the current phase, U uses $cert_U$ as private input. At the end of the protocol V accepts or rejects. Each user executes *Prove* at most once in every phase.
- Algorithm *Detect* gets as input a number of transcripts of executions of *Prove*, done with *pk* as input in the same phase. It outputs a (possibly empty) list of strings. The intuition is that this algorithm should be able to tell if the result of one or more cloning attacks are among a given set of proofs, and if so, it will output the identities of the involved users.

Definition 1. *The algorithms and protocols in a secure unclonable identification scheme must satisfy the following:*

Completeness *Assume GM, V and user U are honest. Execution of $KeyGen$, followed by executions of $Join$ and $Prove$ always result in V accepting.*

No Cloning *Consider an honest GM who executes $(pk, sk) = KeyGen(1^k)$. Consider any probabilistic polynomial time algorithm $\tilde{U}$ who plays the following game on input pk: in any phase, it can issue one or more of the following requests:*

1. *It can ask that a set of honest users execute $Join$ with GM (no data returned to $\tilde{U}$).*
2. *It can ask to execute $Join$ itself with GM.*
3. *It can ask that some number of honest users who already joined the group execute $Prove$ with $\tilde{U}$ acting as verifier, using pk and the current value of α as input.*

Finally, $\tilde{U}$ executes $Prove$ a number of times with an honest verifier V, on input pk and the current value of α.

We now want to capture the idea that in the last step, $\tilde{U}$ can only have proofs accepted by using user identities it got from GM, it must "know" which one of them it is using in each case, and if it uses any of them more then once, the Detect algorithm will catch this.

To this end, we demand that there exists a probabilistic algorithm Extract which gets as input the complete view of $\tilde{U}$[2] *and outputs a user identity, for every instance of $Prove$ that V accepted in the last step. The expected time to run $\tilde{U}$ and then Extract must be polynomial.*

We require that the following holds except with negligible probability:

All user identities output by Extract are among those that were generated in the conversations between $\tilde{U}$ and GM. Furthermore, the Detect algorithm,

[2] This means that *Extract* can rewind $\tilde{U}$ to any state that occurred during the game.

when given as input the conversation between $\tilde{U}$ and V, will output exactly those user identities that occur more than once in the output of Extract.

Note that this implies that if $\tilde{U}$ did not execute any Join's, there are no user identities Extract can legally output, so we are then in fact demanding that all $\tilde{U}$'s proofs are rejected except with negligible probability. Thus we do not need a separate soundness condition in the definition demanding that non-members are rejected.

Anonymity *Consider any probabilistic polynomial time algorithm $\tilde{V}$, who will act as both GM and verifier in an attempt to break the anonymity of honest users. $\tilde{V}$ gets 1^k as input and outputs a valid pk (can be assumed without loss of generality since we assumed that invalid pk's can be easily recognized). It then plays the following game: it interacts with a set of honest users, where in each phase some users execute Join and other users execute Prove with $\tilde{V}$. Of course, no honest user will attempt to do Prove unless he already did Join successfully. At some point $\tilde{V}$ stops and outputs a bit, and we let $p_{real,\tilde{V}}(k)$ be the probability that 1 is output.*

We now want to express the demand that $\tilde{V}$ should only learn what is unavoidable, namely the number of honest users that interact with it in each phase. So we compare the above game to a different one, where $\tilde{V}$ interacts with a simulator M. The simulator gets as input for each phase the number of users who want to execute Join and the number that want to execute Prove in the current phase. These numbers are chosen with the same distribution as in the first game. Let $p_{sim,\tilde{V}}(k)$ be the probability that 1 is output in this case.

We demand that there exists a simulator probabilistic polynomial time simulator M such that for any $\tilde{V}$, $|p_{real,\tilde{V}}(k) - p_{sim,\tilde{V}}(k)|$ is negligible in k.

We note that in this definition, we have for simplicity used the usual two-phase structure of identification schemes to define soundness and non-cloning, where first the adversary talks to the honest users and then tries to fool the honest verifier. Thus we do not allow him to interact with an honest prover and and honest verifier simultaneously. However, this is not a serious restriction, as there are several techniques that allow handling even this concurrent case, such as the so called designated verifier proofs[12, 7]. These techniques can be used with any of the schemes we propose here.

As for the scheduling of the individual protocols in a single phase, we consider two cases: one where in each phase the proofs given to an honest verifier are composed sequentially, and one where the composition may be concurrent, with a scheduling chosen by the adversary. We speak of *sequential* and *concurrent security*, accordingly. On the other hand, we assume that honest users (provers) may interact concurrently with an adversarial verifier.

3 A Theoretical Solution

3.1 Some Tools

We will need a secure string commitment scheme. Such a scheme follows from any one-way function using for instance Naor's construction[18], where there is a public key $Pcom$ which is a random string (of length polynomial in the security parameter k) that can be chosen once and for all by the receiver of commitments. We let $com_{Pcom}(str, r_{str})$ denote a commitment to string str using random coins r_{str}. Such a commitment determines str uniquely except for a negligible fraction of the public keys, and commitments to different strings are polynomially indistinguishable assuming the underlying one-way function is hard to invert.

Based on such a commitment scheme and, for instance, Blum's protocol for Graph Hamiltonicity or the one from [14] for graph 3-colorability, we can build generic proofs of knowledge for any binary relation R that can be checked in polynomial time. The protocol in its basic form is a three move protocol where the second message is a one-bit challenge from the verifier. When we work with security parameter k, we may compose *sequentially* k instances of this protocol, to obtain a zero-knowledge proof of knowledge for R with negligible soundness error. We may also compose *in parallel* k instances of the protocol. This is also a proof of knowledge for R, more precisely, on common input x, the prover proves knowledge of w such that $(x, w) \in R$.

Protocols obtained by this parallel composition are special cases of so-called *Σ-protocols*. By definition, such protocols have three properties: first, conversations are of form (a, e, z), where $a = a(x, w, coins_P)$ is a function of x, w and the prover's random coins, e is a k-bit challenge, and $z = z(x, w, coins_P, e)$ is a function of the prover's private data and the challenge. Based on $x, (a, e, z)$ the verifier decides to accept or reject. Second, the protocol is honest-verifier (computational) zero-knowledge (and is therefore witness indistinguishable). Third, the protocol has the *special soundness property*, i.e., from x and accepting conversations $(a, e, z), (a, e', z')$ with $e \neq e'$, it is easy to compute w such that $(x, w) \in R$.

Using a technique known as the OR-construction[8], one can combine Σ-protocols for two relations R_0, R_1, to obtain a new Σ-protocol, where on input x_0, x_1, the prover proves he knows w such that $(x_0, w) \in R_0$ or $(x_1, w) \in R_1$, without revealing which is the case, i.e., the protocol is witness indistinguishable.

We will need a family of pseudorandom functions[13]. Such a family is indexed by a key s (a random string of length k bits), and can be designed to have any desired (polynomial in k) input and output length, assuming any one-way function. We let $f_s()$ denote such a pseudorandom function. The basic property is that even given oracle access to the function (and not the key), it cannot be efficiently distinguished from a truly random function.

Finally, we will need a secure signature scheme, which can again be built from any one-way function[21]. Such a scheme comes with probabilistic polynomial time algorithms $Gen, Sign, Verify$ for key generation, signing and verifying signatures. $Gen(1^k)$ outputs a key pair $Psign, Ssign$. On input message m and the private key, $Sign$ produces a signature $\sigma = Sign(Ssign, m)$. On input message, signature and public key, $Verify$ produces as output $Verify(Psign, m, \sigma)$ which is *accept* or *reject*.

3.2 The Scheme

We first explain the intuition behind the solution: when joining the group, user U will make a commitment c_U to a random string r_U and will obtain GM's signature σ_U on the commitment. He then proves he is a member of the group by proving that he knows a valid signature σ_U on some message c_U, without revealing either value. Moreover when giving this proof he uses some random coins. These are not chosen at random but pseudorandomly as $f_{r_U}(\alpha)$. That is, he obtains the coins by applying the pseudorandom function to the current α-value, using r_U as key. He also proves that he has done exactly this. Note that this will force a clone of the user to use the same coins if he gives a proof for the same α-value, by security of the commitment and signature schemes. This idea of choosing the randomness for a proof pseudorandomly is somewhat similar to a technique from a completely different context, namely resetable zero-knowledge [15].

The proof given is actually a Σ-protocol, so the transcripts of proofs given by user and clone are of form (a, e, z) and (a', e', z'). But when all inputs and random coins are the same in the two cases, we must have $a = a'$. Furthermore, $e \neq e'$ with overwhelming probability, so if both proofs are accepted, special soundness of the protocol means that one can easily compute the prover's secret, which will immediately identify the user in question.

We now describe the components of our scheme – throughout the descriptions, it is understood that a party who detects an invalid proof or signature will immediately stop and reject:

KeyGen On input 1^k, it generates keys $(Psign, Ssign)$ for the signature scheme and public key $Pcom$ for the commitment scheme (with security parameter k). Finally, it chooses a random k-bit string R. The public key is $pk = (Psign, Pcom, R)$ while the private key is $sk = Ssign$.

Join The user U sends $c_U = commit_{Pcom}(r_U, s_U)$ where r_u is a random k-bit string. GM assigns a unique identity id_U to U, and sends to U a signature $\sigma_U = Sign(Ssign, (c_U, id_U))$ on c_U concatenated by id_U. Also, GM proves in zero-knowledge that he knows a signature (valid under $Psign$) on R. This is easy given that GM knows $Ssign$. The output certificate for U is r_U, s_U, σ_U, id_U, while output for GM is id_U.

Prove Recall that pk and the string α is common input to the protocol. User U first makes commitments C_U, D_U, E_U to c_u, id_U, σ_U, respectively. He will now give a proof of knowledge related to these commitments, the group public key pk and the number α assigned to the current phase. This proof consists of three ingredients. The first is a proof of knowledge, that U knows how to open the commitments C_U, D_U, E_U to strings c_u, id_U, σ_U such that

σ_U is GM's signature on (c_U, id_U). While giving this proof, he uses $f_{r_U}(\alpha)$ as random coins. That is, the protocol transcript is (a_1, e_1, z_1), where it should be the case $a_1 = a_1((pk, C_U, D_U, E_U), (c_U, id_U, \sigma_U), f_{r_U}(\alpha))$.
The second ingredient is a proof that U can open C_U to reveal c_U and he knows s_U, r_U such that $c_U = \mathit{commit}(r_U, s_U)$, and the message a_1 from the previous protocol satisfies $a_1 = a_1((pk, C_U, D_U, E_U), (c_U, id_U, \sigma_U), f_{r_U}(\alpha))$. Also this proof is a three move protocol of form (a_2, e_2, z_2), and we are going to do the two proofs in parallel, so that the overall conversation will have form $(a_1, a_2, e_1, e_2, z_1, z_2)$. The final ingredient is a proof of knowledge of GM's signature on the string R that is part of pk. This is combined with the previous ingredients using the OR construction mentioned above, i.e., U is proving that he knows a signature on R, or strings $c_u, id_U, \sigma_U, r_U, s_U$ satisfying the conditions just described [3].

Detect Looks at all the proofs given in a phase and finds all places where two conversations include tuples of form $(a_1, a_2, e_1, e_2, z_1, z_2)$, respectively $(a_1' a_2', e_1', e_2', z_1', z_2')$ and where $a_1 = a_1'$ and $e_1 \neq e_1'$. For any such case it will use the special soundness property to extract the underlying c_U, id_U, σ_U, and appends id_U to its output list.

Theorem 1. *Assuming one-way functions exist, the above scheme is a secure unclonable identification scheme with sequential security.*

We remark that concurrent security can be obtained under the same assumption in the common reference string model, using a technique similar to the one used in the more efficient protocol we describe later.

The key to the proof of the theorem is

Lemma 1. *The proof of knowledge given by the user during the Prove protocol is witness indistinguishable*

Proof. Recall that the proof given by U is a combination using the OR construction of first a proof of knowledge of a signature on R and second a proof of knowledge of values $c_u, id_U, \sigma_U, r_U, s_U$ satisfying a number of properties. Conversations in the latter protocol are of form $(a_1, a_2, e_1, e_2, z_1, z_2)$. The OR construction leads to a witness-indistinguishable protocol if both protocols used are honest verifier zero-knowledge. This is true for the first protocol, which is just a standard Σ-protocol and so is honest verifier zero-knowledge by construction.

It is therefore enough to show that the second protocol is honest verifier zero-knowledge. Some notation for this: the part (a_1, e_1, z_1) of a conversation will be called *proof 1.* It has the commitments C_U, D_U, E_U and public key pk as public input, while the secret witness is c_U, id_U, σ_U. The rest of the conversation (a_2, e_2, z_2) is called *proof 2.* It has C_U, D_U, E_U, pk, a_1 as public input while the secret witness is $c_U, id_U, \sigma_U, r_U, s_U$.

Both proof 1 and proof 2 are Σ-protocols constructed from generic zero-knowledge techniques as explained above. They therefore have honest verifier

[3] Of course, the latter is normally the case, the other option is included for proof-technical reasons.

simulators M_1, M_2 respectively. However, note that in our context, proof 1 is not done using the normal prover algorithm, we use pseudorandom coins for the prover, and furthermore the key for this pseudorandomness is used as input in proof 2. Hence a proof is required that we can still use M_1, M_2 to simulate. We do this by defining a series of distributions where the first is that of real conversations and the last is the one output by the honest verifier simulator we propose. The result will then follow from arguing that each distribution is computationally indistinguishable from the previous one.

The sequence of distributions are produced as follows:

1. Run the honest prover U's algorithm (with known secret witnesses and random challenges).
2. Same as above, but proof 2 is replaced by running the honest verifier simulator $M_2(C_U, D_U, E_U, pk, a_1)$ for proof 2. Note that this requires that r_U is known, to do proof 1 according to the protocol. However, we will still get something indistinguishable from the previous distribution. This is because the output of M_2 is indistinguishable from a real conversation, *even to someone who knows the secret witness for proof 2*. Indeed, M_2 is simulating a protocol constructed from generic techniques based on any commitment scheme as explained earlier. This means that the simulation essentially produces a set of commitments, some of which are opened and some are not. The unopened commitments have contents different from what would be the case in a real conversation, however, this is the only difference. By the hiding property of the commitments, this difference cannot be detected in polynomial time, even knowing what the commitments are supposed to contain.
3. As 2., but the commitment c_U is replaced by a commitment to a random value. This is indistinguishable from 2. by the hiding property of commitments.
4. As 3., but when doing proof 1, instead of using r_U to compute pseudorandom values for the random coins, we use oracle access to the function $f_{r_U}()$. We now do not know r_u explicitly, but we will produce exactly the same distribution as in 3.
5. As in 4., but the oracle access to $f_{r_U}()$ is replaced by oracle access to a random function. This is indistinguishable from 4. by pseudorandomness of the function $f_{r_U}()$.
6. As in 5., but the transcript of proof 1 is now generated by running the honest verifier simulator M_1 for proof 1. This is indistinguishable from 5., since there, we ran proof 1 following the prover's normal algorithm, using real random coins. Summarizing, this last distribution is generated by first running $M_1(C_U, D_U, E_U, pk)$ to get (a_1, e_1, z_1), and running $M_2(C_U, D_U, E_U, pk, a_1)$ to get (a_2, e_2, z_2), and this defines the desired honest verifier simulation.

We can now proceed with the proof of the required properties.

Anonymity: if $\tilde{V}$ behaves such that at least one instance of the *Join* protocol completes successfully with non-negligible probability, then we can extract from the proof of knowledge given by $\tilde{V}$ a signature on R. Note that no attempts to do *Prove* would occur before this point. Given this signature, it is trivial to simulate (without rewinding) all subsequent instances of *Prove* knowing only the number of instances to be done in each phase. This cannot be distinguished from the real game by witness indistinguishability of the underlying proofs of knowledge.

No cloning: we first describe the required *Extract* algorithm. It will, for each proof $\tilde{U}$ had accepted in the last stage of the attack, rewind $\tilde{U}$ to the start of this proof and try to extract the secret witness it is using by the standard rewinding technique of sending random challenges to $\tilde{U}$ until it answers a new challenge correctly. At this point a valid witness can be extracted. Each such witness must include either a signature on R, or a signature σ_U on a pair of form (c_U, id_U). *Extract* outputs id_U in the latter case, and a random string in the former. We put the limitation that the algorithm gives up on a proof and outputs a random string if it rewinds more than 2^k times, where k is the length of challenges.

To estimate the running time of this, note that the probability that $\tilde{U}$ will have a proof accepted, given the state it is in just before the proof, is determined by the number T of challenges it will answer correctly. The probability that we will have to run *Extract* on the proof is $T2^{-k}$, while the number of rewinds we have to do is 0 if $T = 0$, 2^k if $T = 1$ and $2^k/(T-1)$ if $T > 1$. It follows that contribution to the total expected running time from each proof is polynomial. The total expected running time is just the sum of these contributions since we compose sequentially.

To finalize the argument, we need the following

Claim: we may assume that in the output of *Extract*, we will only see triples (c_U, id_U, σ_U) that were obtained earlier by $\tilde{U}$ in some instance of *Join*.

Indeed, if this is false with non-negligible probability, we can break the signature scheme in a chosen message attack: we choose at random to either ask for signatures on all pairs c_U, id_U or a signature on R and use this to simulate the *Join* protocols done by $\tilde{U}$ and all proofs by honest users given to $\tilde{U}$ (without rewinding, we just follow the protocol). Then by witness indistinguishability, $\tilde{U}$'s behaviour will be essentially the same as before, so the knowledge extraction from $\tilde{U}$ will give us a signature on a new message with non-negligible probability.

Consider now any two *Prove* instances where the same triple c_U, id_U, σ_U is extracted. Let $(a_1, a_2, e_1, e_2, z_1, z_2), (a_1', a_2', e_1', e_2', z_1', z_2')$ be the transcripts of the two instances.

Now, soundness of the *Prove* protocol implies that we could extract (except with negligible probability) also two ways of opening c_U from the two instances, that is, two pairs $(r_U, s_U), (r_U', s_U')$ such that

$$c_U = commit_{Pcom}(r_U, s_U) = commit_{Pcom}(r_U', s_U'),$$

and that

$$a_1 = a(pk, (c_U, id_U, \sigma_U), f_{r_U}(\alpha)), a_1' = a(pk, (c_U, id_U, \sigma_U), f_{r_U'}(\alpha)).$$

But we must have $r_U = r_U'$, or the the binding property of the commitment scheme is broken. This immediately implies that $a_1 = a_1'$, and therefore, since $e_1 \neq e_1'$ with overwhelming probability, *Detect* will successfully extract id_U, as required in the definition.

4 A More Efficient Solution

In this section, we present a more efficient unclonable group identification scheme, based on two main ingredients: First a technique recently proposed by Camenisch and Lysyanskaya [3] for digital signatures based on bilinear groups, with protocols for proving knowledge of a signature on a committed value. Second, a new technique for proving that an element in a group is of form g^{ψ} where ψ is a pseudorandom value computed from a committed key. We will borrow some notation from [3] (and several earlier papers): given a public string x, a private witness w and a predicate *pred*,

$$PK\{w : pred(x, w)\}$$

means that we execute a Σ-protocol for the relation $\{(x, w)| \ pred(x, w) = true\}$, that is, a prover convinces a verifier that he knows w such that the predicate on x and w is satisfied. We will also use the following variant:

$$PK(\kappa)\{w : pred(x, w)\}$$

where κ is a bit string. This stands for the following: we execute the underlying Σ-protocol in the normal interactive way, except that the verifier sends as the second message a random string κ, and the challenge the prover has to answer is determined as $H(x, a, \kappa)$, where H is a hash function, modelled as a random oracle and a is the first message in the original protocol. The point of this construction is that it allows simulation of the protocol without rewinding, due to the "programmability" of the random oracle, and (for the same reason) it also allows knowledge extraction by standard rewinding. Since we will need the last point for the proof, we cannot just use the Fiat-Shamir heuristic.

4.1 Proofs of Knowledge with Pseudorandom Exponents

In this subsection, we introduce some tools to be used in our construction. To this end, we consider a group G_p of prime order p. We will assume p is chosen as a safe prime, i.e., $p = 2q + 1$ where q is also prime. G_q will denote the (unique) subgroup of Z_p^* of order q.

We further consider the case where a prover knows exponents $x_1, ..., x_t \in Z_p$ such that $\beta = \alpha_1^{x_1} \cdots \alpha_t^{x_t}$ for publicly known $\beta, \alpha_1, ..., \alpha_t \in G_p$. A standard Σ-protocol for prover P and verifier V can be used to prove knowledge of the x_i's. That is, we want:

$$PK\{(x_1, ..., x_t) : \beta = \alpha_1^{x_1} \cdots \alpha_t^{x_t}\} \tag{1}$$

This Σ-protocol works as follows:

1. P chooses $r_1, ..., r_t \in Z_p$ uniformly at random and sends to V $\tau = \prod_{i=1}^{t} \alpha_i^{r_i}$.
2. V chooses a random challenge $\epsilon \in Z_p$.
3. P responds with $z_i = r_i + \epsilon x_i \bmod p$ for $i = 1..t$. V checks that $\prod_{i=1}^{t} \alpha_i^{z_i} = \tau\beta^{\epsilon}$.

It is well known (and straightforward to show) that this protocol is indeed a Σ-protocol for the underlying relation. It is also well-known that a straightforward variant of the protocol allows us to do the following type of proof:

$$PK\{(x_1,..,x_t,x'_1,...,x'_t) : \beta = \alpha_1^{x_1}\cdots\alpha_t^{x_t}, \beta' = {\alpha'}_1^{{x'}_1}\cdots{\alpha'}_t^{{x'}_t}, x_1 = x'_1\}$$

Basically, we run the original protocol twice in parallel for the two equations. This would normally involve two independent sets of random numbers $r_1,...,r_t$ and $r'_1,...,r'_t$. However, to demonstrate that $x_1 = x'_1$ the prover must use $r_1 = r'_1$ and the verifier checks that the responses $z_1,...,z_t,z'_1,...z'_t$ satisfy $z_1 = z'_1$. We will use this variant later, but for now we stick to the basic version for simplicity. All techniques we describe here can also be applied to the variant in a straightforward way.

We now consider a change to the protocol where P chooses the randomness in the first message according to a pseudorandom function $\Psi_K(i,\alpha,b)$, where K is a key committed to by P, α is a public input, i is a number and b is a bit. We will use a variant of the pseudorandom function of Naor and Reingold, based on the DDH assumption in G_q, so that outputs from Ψ are in G_q. We specify below how the function works and how the key is committed. However, in the previous protocol, the random exponents were chosen in Z_p, whereas the pseudorandom function produces output in the subgroup G_q. To resolve this, we let the exponents be chosen as the difference between two pseudorandom values, which allows us to hit all of Z_p. The modified protocol then works as follows:

1. P sets $r_i = \Psi_K(i,\alpha,0)$ and $s_i = \Psi_K(i,\alpha,1)$ and sends to V $\tau = \prod_{i=1}^t \alpha_i^{r_i-s_i}$.
2. V chooses a random challenge $\epsilon \in Z_p$.
3. P responds with $z_i = r_i - s_i + \epsilon x_i \bmod p$ for $i = 1..t$. V checks that $\prod_{i=1}^t \alpha_i^{z_i} = \tau\beta^\epsilon$.

To argue that this is a Σ-protocol for the same relation, we need a result by Perron[20]: Let QR_p be the set of quadratic residues mod p. Then for any $a \in Z_p*$, the set $a + QR_p$ contains almost as many quadratic residues as non-residues: the difference is at most 1. Since in our case $G_q = QR_p$, we get from this:

Lemma 2. *The distribution of $u_i - v_i \bmod p$ where u_i, v_i are chosen uniformly in G_q, is statistically close to uniform over Z_p.*

Lemma 3. *Under the DDH assumption in G_q, the above protocol is a Σ-protocol for the relation specified in (1).*

Proof. Completeness is trivial, and special soundness follows exactly as for the previous standard protocol. For honest verifier zero-knowledge, we argue as follows: To simulate, we will choose ϵ and z_i at random in their respective domains and then set $\tau = \beta^{-\epsilon}\prod_{i=1}^t \alpha_i^{z_i}$.

Now, assuming K is known only to P, pseudorandomness of Ψ implies that our variant is indistinguishable from a protocol where $\Psi_K(i,\alpha,0), \Psi_K(i,\alpha,1)$ are replaced by uniformly random choice u_i, v_i from G_q. This creates a distribution of z_i that is statistically close to the simulated distribution by Lemma 2.

Our goal is now to allow P to prove that he has followed the specified algorithm for choosing the r_i, s_i's pseudorandomly. The first step of this is to have P commit to each individual value under a public key chosen by a third party (which will eventually be the group manager in our case). The public key will be two random elements $\eta, \lambda \in G_p$, and P will make commitments $com_i = \eta^{r_i}\lambda^{\omega_i}$ and $com'_i = \eta^{s_i}\lambda^{\omega'_i}$, for $i = 1..t$ and random ω_i, ω'_i. We can now ask P to prove that he committed to the correct values, that is, execute

$$PK\{(r_i, s_i, \omega_i, \omega'_i, i = 1..t) : \tau = \prod_{i=1}^{t} \alpha^{r_i}(\alpha^{-1})^{s_i}, \\ com_i = \eta^{r_i}\lambda^{\omega_i}, com'_i = \eta^{s_i}\lambda^{\omega'_i}, i = 1..t\}$$

The Σ-protocol for this is a standard variant of the one we presented above.

The final step is to show that each committed value was chosen according to the pseudorandom function. For this, we need to specify in detail how it works. We assume that input strings to Ψ all have length at most k (where k can in principle be arbitrary). A key to the function is a number $K \in Z_q$. Finally, we will need a hash function H that take a string str of length at most k as input and outputs an element in G_q. We will model this function as a random oracle. The pseudorandom function is now defined as:

$$\Psi_K(str) = H(str)^K \bmod p$$

We note that the function mapping y to $y^K \bmod p$ is a weak pseudorandom function assuming the DDH assumption holds in G_q, i.e., as long as y is randomly chosen and is not controlled by the adversary, the outputs look random. However, in our case, and assuming the random oracle model, the function is only used on values produced by H, and these are guaranteed to be random, even if the adversary chooses the inputs to H. This argument is easily formalized to prove

Lemma 4. *In the random oracle model, and assuming DDH holds in G_q, $\Psi_K()$ as defined above is a strong pseudorandom function.*

We will assume that the key K is committed to by P in a somewhat non-standard way which, however, fits nicely with the construction we will see in the following. Concretely, we assume that $d = g^{\gamma^K \delta^r} h^u$ is given, for publicly known $g, h \in G_p$ and $\gamma, \delta \in G_q$. With this, we can summarize our goal, namely to give a Σ-protocol implementing

$$PK\{(K, r, u, \omega_i, \omega'_i, i = 1..t) : d = g^{\gamma^K \delta^r} h^u, \\ com_i = \eta^{\Psi_K(i,\alpha,0)}\lambda^{\omega_i}, com'_i = \eta^{\Psi_K(i,\alpha,1)}\lambda^{\omega'_i}, i = 1..t\}$$

For this, it will be be enough to show how P can prove that some given commitment com satisfies $com = \eta^{\Psi_K(str)}\lambda^{\omega}$ for public str. Since anyone can compute $\psi = H(str)$, our task reduces to:

$$PK\{(K, r, u, \omega) : d = g^{\gamma^K \delta^r} h^u,\ com = \eta^{\psi^K}\lambda^{\omega}\} \tag{2}$$

A protocol for this follows here:

1. P chooses $s, w \in Z_q, \nu, \phi \in Z_p$ at random. He sends $v_1 = g^{\gamma^s \delta^w} h^\nu$ and $v_2 = \eta^{\psi^s} \lambda^\phi$ to V.
2. V selects a random bit c.
3. P sends $z_1 = s - cK \bmod q$, $z_2 = w - cr \bmod q$, $z_3 = \nu - cu\gamma^{s-K}\delta^{w-r} \bmod p$, and $z_4 = \phi - c\omega\psi^{s-K} \bmod p$.
   V checks as follows: if $c = 0$, that $g^{\gamma^{z_1}\delta^{z_2}} h^{z_3} = v_1$ and $\eta^{\psi^{z_1}} \lambda^{z_4} = v_2$. If $c = 1$, that $d^{\gamma^{z_1}\delta^{z_2}} h^{z_3} = v_1$ and $com^{\psi^{z_1}} \lambda^{z_4} = v_2$.

Since this protocol only works with a 1-bit challenge, we need to repeat it an appropriate number of times to have a sufficiently small soundness error.

Lemma 5. *The above is a Σ-protocol for the relation specified in (2)*

Proof. Completeness follows by inspection of the protocol. Special soundness: if for given v_1, v_2, the prover can send satisfactory answers z_1, z_2, z_3, z_4 to $c = 0$ and z_1', z_2', z_3', z_4' to $c = 1$, we have by the checks carried out by V that $g^{\gamma^{z_1}\delta^{z_2}} h^{z_3} = v_1$, $\eta^{\psi^{z_1}} \lambda^{z_4} = v_2$. $d^{\gamma^{z_1'}\delta^{z_2'}} h^{z_3'} = v_1$ and $com^{\psi^{z_1'}} \lambda^{z_4'} = v_2$. Combining these equations imply that $com = \eta^{\Psi^{z_1 - z_1'}} \lambda^{(z_4 - z_4')\psi^{-z_1'}}$ and $d = \alpha^{\gamma^{z_1 - z_1'}\delta^{z_2 - z_2'}} h^{(z_3 - z_3')\gamma^{-z_1'}\delta^{-z_2'}}$, i.e., a, d are of the required form. Finally, honest verifier ZK is argued by the following simulator: choose z_1, z_2 at random in Z_q, z_3, z_4 at random in Z_p and c as a random bit. If $c = 0$, set $v_1 = g^{\gamma^{z_1}\delta^{z_2}} h^{z_3}$ and $v_2 = \eta^{\psi^{z_1}} \lambda^{z_4}$. If $c = 1$, set $v_1 = d^{\gamma^{z_1}\delta^{z_2}} h^{z_3}$ and $v_2 = com^{\psi^{z_1}} \lambda^{z_4}$. This simulation is seen to be perfect by a standard argument.

4.2 The New Scheme

Our main idea for the scheme is similar to the earlier theoretical one: the user U will commit to a secret key K. When registering with the group manager GM he will obtain a signature on the commitment c_U, using the signature system described in [3] (called scheme A in [3]). He can now prove membership of the group by proving knowledge of a valid signature on c_U (as well as proving knowledge of this value). If he tries to clone his identity we can exploit the special soundness property of the protocol used and extract his identity.

KeyGen Let GM take a security parameter k and output two groups $G_p = \langle g \rangle$ and $\mathbf{G_p} = \langle \mathbf{g} \rangle$ of prime order $p = \Theta\left(2^k\right)$ where $p = 2q + 1$ and q is a prime. Let G_q denote the unique subgroup of Z_p^* of order q. Let γ, δ be random generators of G_q. Let $e : G_p \times G_p \to \mathbf{G_p}$ be an efficiently computable bilinear map, and η, λ be random generators of $\mathbf{G_p}$.
To set up the signature scheme, GM chooses the following values at random: $x \in Z_p$, $y \in Z_p$ and sets $X = g^x$, $Y = g^y$. The secret key for the signature scheme is $S_k = (x, y)$ and the public key is $P_k = (q, G_p, \mathbf{G_p}, g, \mathbf{g}, e, X, Y, \eta, \lambda)$.

Join The user U chooses at random $r_U \in Z_q$ and a key $K \in Z_q$. U makes a commitment $c_U = \gamma^K \delta^{r_U} \bmod p$ to K and sends it to GM[4].

[4] It is not necessary to have U prove that he can open c_U, since later in the *Prove* protocol, he must implicitly show he can open it to have the proof accepted.

GM verifies that U is allowed to join the group and if so, he computes a signature $\sigma = (a, b, c)$ on c_U where a is chosen at random in G_p, $b = a^y$, $c = a^{x+c_U xy}$ and sends it to U. GM considers c_U as the user's id in the following, whereas (K, r_U, a, b, c) serves as the membership certificate.

Prove Recall that the string α, denoting "the current time", is common input to U and V. U essentially proves that he is a member of a group by proving that he knows a valid message and signature from GM. First U blinds his signature σ by choosing at random $\mu, r' \in Z_p$ and computing $\widetilde{\sigma} = \left(\widetilde{a}, \widetilde{b}, \hat{c}\right)$ where $\widetilde{a} = a^{r'}$, $\widetilde{b} = b^{r'}$, $\hat{c} = (c^{r'})^{\mu}$. Then U makes a commitment $C_U = \eta^{c_U} \lambda^{s_U}$ for random s_U and sends $\widetilde{\sigma}$ and C_U to V. Both compute

$$v_x = e\left(X, \widetilde{a}\right), \quad v_{xy} = e\left(X, \widetilde{b}\right), \quad v_s = e\left(g, \hat{c}\right)$$

V chooses a k-bit string κ at random, and U proves knowledge of a signature on c_U to V by giving the following proof:

$$PK(\kappa)\{(c_U, \rho, s_U) : C_U = \eta^{c_U} \lambda^{s_U}, {v_s}^{\rho} = v_x v_{xy}^{c_U}\} \qquad (3)$$

Here, as ρ, the honest U uses $\rho = \mu^{-1} \bmod p$. V will accept if this proof is correct and it holds that:

$$e\left(\widetilde{a}, Y\right) = e\left(g, \widetilde{b}\right)$$

Note that it was shown in [3] that the checks carried out by V plus the proof that ${v_s}^{\rho} = v_x v_{xy}^{c_U}$ together imply that U must know a valid signature on a message. Doing the proof in (3) is a straightforward variant of the general type of proof from lemma 3 as discussed earlier. Consider the underlying Σ-protocol for the part relating to ${v_s}^{\rho} = v_x v_{xy}^{c_U}$. After specializing it to the concrete scenario here, it will have a first message of form

$$\tau = v_{xy}^{r_1 - s_1} v_s^{r_2 - s_2},$$

Furthermore, we will require that

$$r_1 = \Psi_K(1, \alpha, 0), r_2 = \Psi_K(2, \alpha, 0), s_1 = \Psi_K(1, \alpha, 1), s_2 = \Psi_K(2, \alpha, 1)$$

U must therefore prove that the values of r_1, r_2 and s_1, s_2 were generated pseudorandomly from K. As described earlier, U does this by making commitments

$$com_1 = \eta^{r_1} \lambda^{\omega_1}, com_2 = \eta^{r_2} \lambda^{\omega_2}, com'_1 = \eta^{s_1} \lambda^{\omega'_1}, com'_2 = \eta^{s_2} \lambda^{\omega'_2},$$

proving that these values are correct with respect to τ and finally using the protocol from Lemma 5 to show that each commitment contains a pseudorandom value of correct form computed from K as committed to by C_U. Note that C_U is in fact a commitment to K of exactly the form needed for that protocol.

All proofs to be given during Prove can be done simultaneously, using the same challenge in all Σ-protocols.

The amount of computation required during Prove is $57 + 68r$ exponentiations and 8 evaluations of the function e, where r is the number of times the protocol from Lemma 5 is repeated. Prove also requires $29k + r(6k + 1)$ bits sent between U and GM, where r is the same as before and k is the security parameter. For example with $r = 16$ and $k = 1024$, Prove requires approximately 1150 exponentiations and needs to communicate $130KB$. It would be interesting to find a protocol that solves the same problem using a constant number of exponentiations and communication.

Detect Look at all proofs given in a phase and find all places where two conversations include first messages τ, τ' where $\tau = \tau'$. If the two challenge values involved in these two conversations are different, use the special soundness property to extract a witness for the proof in question - this will be a pair of form (c_U, ρ). Output all c_U's found this way.

Theorem 2. *Assuming security of the signature scheme from [3], the DDH assumption in G_q, and Assumption 1, the scheme described above is a secure unclonable identification scheme in the random oracle model, with sequential security. The Join and Prove protocols are constant-round, and have communication complexity $O(k)$ bits, respectively $O(k^2)$ bits.*

The scheme described here is extremely similar in structure to the theoretical solution we gave earlier, so the proof is very similar as well. We only sketch it here. Completeness follows by inspection of the protocols. For no cloning, the required *Extract* algorithm will use standard rewinding to extract witnesses for all proofs given. By a standard argument, this will succeed for all proofs that were accepted by the verifier, with overwhelming probability. Soundness of the proofs means we will extract a set of user id's and corresponding signatures, so security of the signature scheme implies that this forms a subset of the user id's defined in previous *Join* protocols. Now, soundness of the proofs from Lemma 5 and the binding property of the commitment schemes defined by $(\gamma, \delta), (\eta, \lambda)$ imply that the adversary must have used the key involved correctly and consistently, and hence the value of τ will be identical in all instances of subproof (3), where the same key was used. This allows *Detect* to recover the required information. As for anonymity, note that all subproofs except the one from (3) can be replaced by (perfect) simulations without changing the view of the adversary. After this change, the key K is only used to call the pseudorandom function, and no other information on K is present, since the commitment c_U hides K perfectly. We can therefore use Lemma 3 to conclude that also instances of subproofs from (3) can be replaced by simulations without this being detectable by the adversary.

5 On Concurrent Security

For both the theoretical and the more efficient solution, it holds that all the proofs given by honest users can be simulated without rewinding. Hence, the only problem in obtaining concurrent security lies in the *Extract* algorithm that is required for the no cloning property, and which requires rewinding in both solutions.

To avoid this, we can use, at a small efficiency cost, the technique of Fischlin [11], which shows how to transform any Σ-protocol in the random oracle model into a new one for which there is an on-line extractor, i.e., one can extract the secret witness from a successful prover without rewinding.

Using this transformation on the Σ-protocols underlying our *Prove*-protocol immediately gives a concurrently secure solution.

6 On membership revocation and framing

After discovering the identity of a dishonest user, the group manager needs to act. In some applications it may be sufficient to take some appropriate, say, legal action against the user in question. But it may also be necessary to remove the user out from the group by ensuring that the value c_U can never be used again.

Since the value c_U is unconditionally hidden in the *Prove* protocol, nothing in the above systems prevents a dishonest user from proving membership of the group again at a later point in time. To allow for revocation of memberships, we can extend the protocol with an *dynamic accumulator* as described in [4]. An *accumulator scheme* [1, 2] is an algorithm that allows one to hash large set of values into a short value, called the *accumulator* such that there is a witness that a given input is in the accumulator. A *dynamic accumulator* allows one to efficiently add and remove values from the accumulator. It can be used in the following way.

When the user joins the group and sends c_U, the group manager adds c_U to the accumulator. To prove membership of the group, the user is now required, in addition to the protocol we already have, to prove that the value c_U is in the accumulator. We will omit the details of how this is done, they can be found in [4]. The solutions needs that c_U is committed to, but this is already done in our protocol.

When the identity of a dishonest user is discovered, the group manager removes c_U from the accumulator, which prevents the user or any clones of the user from proving membership of the group, as long as the verifier is aware of the new accumulator value.

An aspect we have not been concerned with in this paper is whether the group manager can *frame* an honest user, that is, create on his own a protocol transcript where the user seems to have cloned himself. For our efficient solution, we believe framing is not possible, since the group manager does not know the user's secret key K. The part of the *Prove* protocol where the user proves knowledge of K can only be simulated without knowing K if one can control the outputs of the random oracle. While we use this to show zero-knowledge in the theoretical analysis, such control is not available to anyone in real life.

References

1. Josh Benaloh, Michael de Mare: *One-Way Accumulators: A Decentralized Alternative To Digital Signatures*, proc. of EUROCRYPT 1993.
2. Josh Benaloh, Michael de Mare: *Collision-free accumulators and fail-stop signature schemes without trees*, proc. of EUROCRYPT 1997.
3. J. Camenisch, A. Lysyanskaya: *Signature Schemes and Anonymous Credentials from Bilinear Maps*, Proc. of Crypto 04, Springer Verlag LNCS 3152.
4. J. Camenisch, A. Lysyanskaya: *Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials*, Proc. of Crypto 02, Springer Verlag LNCS 2442.
5. G.Ateniese, J.Camenisch, M.Joye, G.Tsudik: *A practical and provably group signature scheme*, Proc. of Crypto 00, Springer Verlag LNCS 1880.
6. S.Brands: *Untraceable Off-line Cash in Wallets with Observers*, proc. of Crypto 93.
7. Ronald Cramer, Ivan Damgård: *Fast and Secure Immunization Against Adaptive Man-in-the-Middle Impersonation.* EUROCRYPT 1997: 75-87
8. Ronald Cramer, Ivan Damgård, Berry Schoenmakers: *Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols*, CRYPTO 1994: 174-187
9. Ivan Damgård, Mads Jurik: *Client/Server Tradeoffs for Online Elections.* Public Key Cryptography 2002: 125-140
10. Damgård, Dupont and Pedersen: *Unclonable Group Identification* the Eprint archive, www.iacr.org.
11. M.Fischlin: *Communication-Efficient Non-Interactive Proofs of Knowledge*, proc. of Crypto 2005, Springer Verlag LNCS 3621.
12. Markus Jakobsson, Kazue Sako, Russell Impagliazzo: *Designated Verifier Proofs and Their Applications.* EUROCRYPT 1996: 143-154.
13. Oded Goldreich, Shafi Goldwasser, Silvio Micali: *How to Construct Random Functions*, FOCS 1984: 464-479.
14. Oded Goldreich, Silvio Micali, Avi Wigderson: *Proofs that Yield Nothing But Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems* J. ACM 38(3): 691-729 (1991).
15. Ran Canetti, Oded Goldreich, Shafi Goldwasser, Silvio Micali: *Resettable zero-knowledge* (extended abstract). STOC 2000: 235-244
16. J.Kilian and E.Petrank: *Identity Escrow*, Proc. of Crypto 98.
17. D.Chaum, A.Fiat, M.Naor: *Untraceable Electronic Cash*, proc. of CRYPTO 88.
18. Moni Naor *Bit Commitment Using Pseudorandomness*, J. Cryptology 4(2): 151-158 (1991).
19. Pascal Paillier: *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes.* EUROCRYPT 1999: 223-238.
20. Perron: *Bemerkungen über die Verteilung der quadratischen Reste*, Math.Z. 56 (1952), pp.122-130.
21. John Rompel: *One-Way Functions are Necessary and Sufficient for Secure Signatures*, STOC 1990: 387-394.
22. Aggelos Kiayias, Moti Yung: *Group Signatures with Efficient Concurrent Join.* EUROCRYPT 2005: 198-214