Time-efficient Non-interactive Session-key Distribution with Distributed Personal-key Generation

Proposed for presentation at Africacrypt 2012

http://www.aui.ma/old/africacrypt2012/en/index09c3.html?id=116

Abstract. Presented is a time-efficient scheme for non-interactive (session) key distribution (NIKD) where the generation of the personal (long-term) keys is distributed. In this way, single points of vulnerability and the necessity of a single Big Brother-like authority are avoided. The scheme is based on a new method for the construction of identity-based cryptosystems: identity-determined factors selection. This method enables schemes with a considerably reduced computational complexity of the generation of the shared (session) key. Thus, the scheme is particularly suited for execution environments with limited computation capabilities, e.g., mobile or embedded devices. The computational complexity of the generation of a shared key is for today's typical values of the security parameter k (128 or higher) less than 60^{-1} of what the most time-efficient existing practical NIKD schemes can achieve. The process requires on each side only one exponentiation (of a fixed base) in a first algebraic group plus (at most) $\approx k$ multiplications in a second group. The first group can be an ordinary elliptic-curve group of order and field size $2k$ bits long. The second group is a subgroup of the multiplicative group of a prime field where the modulus is only $2k$ bits long. The computational cost of the generation of the personal keys remains on all sides modest, i.e., practical.

Keywords. Identity-based Cryptography, Non-interactive key distribution, Distributed personal-key generation, Identity-determined factors selection

1 Introduction

A system for non-interactive key distribution enables the users of the system to establish pairwise-shared keys without any exchange of information. The latter includes that there is no need for any party to retrieve in advance a public key or public-key certificate of the other party. The only information each party needs (besides the system parameters) is its personal (long-term) key and the identity of the other party. Thus, non-interactive key distribution (NIKD) is necessarily identity-based, each user in the system must have a unique identity (the "user" may be any type of entity with an own identity). Further, each user must have a (personal) private key. An NIKD scheme *cannot* enable a single user to generate her private key completely just by herself because that would also enable her to generate the private keys of *other* users and, using their "private" keys, enable her to generate the keys *they* "exclusively share" with other users. Instead, the generation of a user's private key unavoidably requires the use of one or more *master keys* which are *not known to the user* but only to some trusted authorities that contribute to the generation of the users' private keys.

NIKD is a useful building block for the construction of a variety of cryptographic protocols. Examples are (i) systems for identity-based encryption where the fact that not only the recipient but also the sender must, besides the (public) system parameters, use her own personal key is not a hindrance or is even necessary, e.g., for identity-based *authenticated* encryption (see, e.g., [19]), and (ii) key agreement protocols, most notably for *authenticated* session-key agreement, where the number of protocol messages is reduced by making these protocols identity-based (see, e.g., [33] [9]).

In many potential application environments of NIKD, most notably, smart cards and embedded and/or mobile computing devices, the computing power of the devices on which the shared keys (e.g., message keys, session keys) are generated is tightly limited. The combination of scarce computing capabilities and medium-to-high security levels calls for NIKD solutions where the computational complexity of the shared-key generation is kept to a minimum. Another worry related to identity-based key agreement such as NIKD is that the secrecy of the private keys of all users (thus, the security of all communications) depends on the secrecy of *one* master key and the integrity of the Trusted Authority (TA) that generates the personal keys using this master key. These are the two problems for which this paper proposes a solution. In order to address the first problem, we introduce a new approach for the construction of identity-based protocols: *identity-determined factors selection* (IDFS). In order to address the second problem, we introduce the notion of non-interactive key distribution *with distributed private-key generation* (NIKD-DPKG). NIKD-DPKG is a generalization of NIKD. NIKD is the special case of NIKD-DPKG where the number of TAs is 1.

After having decided for the elliptic-curve/discrete-log based approach as the evident path to minimum computational complexity for medium-to-high security levels, an obvious way to minimum complexity is a triad of design choices: (i) avoiding in the shared-key generation algorithm the most costly types of operation, most importantly (bilinear) pairings, but also general (i.e., not fixed-base) exponentiations (point multiplications), (ii) choosing algebraic groups with group operations of minimum computational cost, i.e., (a) where the generalized discrete-log problem (GDLP, see e.g., [22]) in the group must be hard, choosing ordinary elliptic-curve groups with high embedding degree and small field sizes, and, (b) where GDLP hardness is not needed, choosing a simple subgroup of the multiplicative group of a prime field with small field size[1], and (iii) generally minimizing the number of such group operations that require substantial computation.

This paper presents a scheme for *non-interactive key distribution with distributed private-key generation* (NIKD-DPKG) whose design follows from these three choices. In this scheme, the computational complexity of the generation of a shared key is essentially just one fixed-base exponentiation (point multiplication, in the following always just called exponentiation[2]) in an ordinary prime-order elliptic-curve (EC) group (not different from what one may choose for EC-based *standard public-key* cryptography for the same security parameter k) plus (at most) $\approx k$ inexpensive modular multiplications where the modulus is just $2k$ bits long. The computational effort that a TA must spend for enabling a personal key is moderate, essentially $2k+6$ fixed-base exponentiations in a prime field with a modulus $2k$ bits long, plus a fixed-base exponentiation in the EC-group. The size of the personal keys is substantial and must in environments with very tight storage capabilities or where the bandwidth of the link between TA and user is very small be carefully considered, it is typically $(4k+14)k$ bits.

The paper is organized as follows: The rest of this section discusses related work and points out the contributions of this paper. Section 2 introduces the fundamental (new) concept of *identity-determined factors selection* (IDFS) in an intuitive way through the informal presentation of the conceptual development of our IDFS-based NIKD-DPKG scheme. Section 3 gives a formal definition of NIKD-DPKG and its security. Section 4 then presents our IDFS-based NIKD-DPKG scheme in detail. The section details the scheme's five algorithms, shows their consistency, and shows the security of the scheme. Section 5 concludes the presentation with a comparison of the efficiency of the scheme with typical public-key encryption schemes.

1.1 Related Work

Concerning the search for solutions for *non-interactive* key distribution (NIKD), the ball was kicked-off in 1984, by Shamir's proposal of identity-based cryptography (IBC) [32], and in particular by his call for schemes for identity-based encryption (IBE). In essence, most early approaches to IBE were actually NIKD schemes which required the use of a personal (long-term) key not only for the decryption but also for the encryption., e.g., [26].

The first NIKD scheme which did not already per definition impose impractical constraints on the usage environment (such as that users do not collude, or that the personal (private) key must be kept secret from its owner) was presented by Maurer and Yacobi in 1991 [21]. The scheme makes use of special RSA moduli to construct groups in which knowledge of some trapdoor information makes DLP in the group easier to solve than when the trapdoor is not available. Although also the Maurer-Yacobi scheme has meanwhile turned out not to be secure in today's usual security models (see, e.g., [23]), it has spawned the development of a series of other NIKD schemes which use trapdoor discrete log (TDL) groups, e.g., [23], most recently one by Paterson and Srinivasan [27]. An evident performance problem of all known schemes that make use of TDL groups is that the relatively small difference between the computational complexities for generating a private key and for breaking the security of the scheme leads to an extremely high computational load at the side of the TA. Given the current state of the search for suitable TDL groups, the approach is for security levels of 128 and higher not yet practical.

The breakthrough to practical NIKD and IBE were the first *pairing*-based NIKD schemes and IBE schemes by Sakai, Ohgishi, and Kasahara [29] and by Boneh and Franklin [5] around the turn of the millenium. Following this path, a series of pairing-based NIKD schemes was presented in the following years, including proposals based on a formal security model, e.g., [11]. The most encompassing set of features and characteristics, including hierarchical identity spaces, is achieved by [14]. A problem with all theses schemes in environments with devices of limited computing power is the comparatively high computational complexity of the generation of the shared key. In contrast to some pairing-based *IBE* and *identity-based key-encapsulation* schemes (see, e.g., [4] [8] [3], for surveys [6] [16]), all reasonably secure pairing-based *NIKD* schemes presented until today require for the generation of the shared key the computation of a pairing, at least one, on each side. Although the computational costs of the other computations, e.g., exponentiations and cryptographic hashes into elliptic-curve groups, are also significant, the dominant computational cost is the cost of the pairing(s). Thus, we focus in our comparison on this particular cost. For a security parameter $k = 128$, the cost of a pairing can (even optimistically) not be expected to be less than

[1] Compare with the construction of Schnorr signatures [31] and, following up, DSA and ECDSA [24].

[2] Generic groups are written multiplicatively in this paper.

the cost of 20 regular (not fixed-base) exponentiations in the same elliptic-curve group[3]. For higher values of k, the factor rises steeply (see, e.g., [18]). The apparent necessity in pairing-based key-agreement protocols to compute pairings at runtime (for generating a shared key) combined with the high computational cost of these pairings is the main motivation for the continuous work on pairing-free approaches to computation-efficient identity-based (authenticated) key agreement protocols (see, e.g., [12]). However, we are not aware of fully practical results in the branch of *non-interactive* key agreement.

The first formal model of the security of NIKD schemes was presented by Dupont and Enge [11]. Our definitions of NIKD-DPKG and its security extend the definitions of (single-authority) NIKD schemes (there called ID-NIKD schemes) and their security given by Paterson and Srinivasan [27]. Their security model extends Dupont and Enge's in that it *directly* recognizes that an attacker may learn the key shared by two users (or even many shared keys) but this shall still not enable the attacker to learn anything about the shared key of any other pair of users. Furthermore, Paterson and Srinivasan introduce the notion of indistinguishability (of the shared keys) into NIKD. Our security model extends Paterson and Srinivasan's model in that it provides for a multitude of authorities and for distributed personal-key generation.

1.2 Contributions

In the NIKD scheme presented, the preparation of the personal keys is distributed over arbitrarily many independent authorities and computing platforms (NIKD-DPKG), so avoiding single points of vulnerability and the necessity to give Big Brother-like powers to a single TA. The computational complexity of the generation of a shared key is for usual values of the security parameter (128 and higher) less than $\frac{1}{60}$ of today's practical (pairing-based) NIKD schemes while the computational complexity of the generation of the personal keys remains (in contrast to existing TDL-based NIKD schemes) feasible in all kinds of computing environments. Comparing the complexities for all cases precisely is difficult because of the big variety of instantiations of the pairing-based schemes with concrete elliptic curves. Thus, the value $\frac{1}{60}$ is just the result of a rough (optimistic, when looking from a pairing angle) calculation: Following Galindo and Garcia [13], a pairing costs more than 20 regular exponentiations. A regular exponentiation usually means, on average, $1.5q$ group operations (see, e.g., [15] or [22]) where q is the order of the group. This means that the pairing alone in a pairing-based shared-key generation costs as much as $30q$ group operations. As we will see, the generation of a shared-key in our NIKD-DPKG scheme requires (as the computationally dominant computation) one fixed-base exponentiation, in an ordinary elliptic-curve group of high embedding degree and thus typically smaller field size (however, in our rough comparison here assumed to be equal). A fixed-base exponentiation usually means, on average, $0.5q$ group operations (see, e.g., [15] or [22]). Hence, the ratio of the dominant costs is $\frac{0.5q}{1.5q \cdot 20} = \frac{1}{60}$. This calculation does not consider the cost of the other operations. Looking at these, it is reasonable to assume that they are higher on the pairings side (multiple pairings in some schemes, additional exponentiations in others, and cryptographic hashes onto elliptic-curve groups must all be expected to cost more than modular multiplications in prime fields of small size).

The scheme's fundamental method, called *identity-based factors selection*, is new and may have other applications in identity-based cryptography as well. It is an alternative to paring-based approaches that is time-efficient at all sides, the users' side and the Trusted Authorities' side.

2 Identity-determined Factors Selection – an Intuitive Introduction

In discrete-logarithm based public-key cryptography, one typically first specifies an algebraic group G_p of (usually prime) order p and a generator α of this group, and then publishes them as parts of the public key of the respective user, or as public system parameters which are the same for all users' public keys (the latter case dealt with here). The group G_p is typically chosen such that the average complexity of the most efficient algorithm to solve the discrete logarithm problem (see, e.g., [22]) in G_p is $\geq 2^k$ where k is the security parameter. The private key ω_{A} of a user A is randomly chosen from $\mathbb{Z}_p^*$, and $y_{\mathsf{A}} \in G_p = \alpha^{\omega_{\mathsf{A}}}$ is made available to the other users as A's public key.

The ancestor of all discrete-logarithm based public-key cryptography is the Diffie-Hellman protocol (see [10], or, e.g., [22]). Let's say we want to make the Diffie-Hellman protocol (in the subgroup of order p of the multiplicative group of $\mathbb{F}_q$ where q and p are both prime numbers) identity-based and non-interactive.

Aiming at an identity-based and non-interactive version of the Diffie-Hellman protocol, the crucial question can be asked like this: How can a user B who wants to securely communicate with user A retrieve A's public key y_{A} ($= \alpha^{\omega_{\mathsf{A}}} \mod q$) without any interaction with A, or querying a public-key directory (or the like) before the actual communication? Likewise, how can A retrieve B's public key y_{B} ($= \alpha^{\omega_{\mathsf{B}}} \mod q$) without any interaction?

[3] Compare, for instance, with Galindo and Garcia's cost estimate in [13].

The raw idea of *identity-determined factors selection* (IDFS) is:
The Trusted Authority TA (which is always needed for identity-based cryptography (IBC) or NIKD, see, e.g., [32] [21] [5] [3] [11] [27]) selects G_p (i.e., q, p) and α. As its *master key*, the TA creates an indexed array (for short: *vector*) $\mathbf{e}$ of elements randomly chosen from $\mathbb{Z}_p$. The number of elements z is such that $z \geq 2k$, which means, z is big enough such that the average complexity to find a collision of a cryptographic hash function of output length z is not smaller than 2^k. From the (secret) vector $\mathbf{e}$, the TA computes another vector of z elements (as for $\mathbf{e}$, the index runs from 1 to z). The elements of this vector $\mathbf{f}$ are in G_p. The TA computes them as follows: $\forall i \in \{1...z\}: \ \mathbf{f}[i] \leftarrow \alpha^{\mathbf{e}[i]} \mod q$. Together with G_p (i.e., q and p) and α, the TA publishes $\mathbf{f}$ as a public system parameter.

As typical in IBC and NIKD, user A's private key ω_{A} is generated by the TA, and ω_{A} corresponds to A's identity, a bit string $\mathsf{ID_A}$ which is unique in the system. The TA first computes a cryptographic hash of the identity: $\mathbf{s}_{\mathsf{ID_A}} \in \{0,1\}^z \ \leftarrow \ \mathrm{hid}(\mathsf{ID_A})$ (*hid* is a standard cryptographic hash function, such as SHA [25], and publicly known as part of the system parameters). Then the TA computes ω_{A}. For this modular *addition*, the z elements of the bit vector $\mathbf{s}_{\mathsf{ID_A}}$ (the index running from 1 to z) select the summands from a set (precisely, a vector) of z *potential* summands, namely from $\mathbf{e}$. Concretely, the TA computes $\omega_{\mathsf{A}} \in \mathbb{Z}_p \ \leftarrow \ \sum_{i=1}^{z} (\mathbf{s}_{\mathsf{ID_A}}[i] \ \cdot \ \mathbf{e}[i]) \mod p$ (in compact *scalar product* notation: $\omega_{\mathsf{A}} \ \leftarrow \ \mathbf{s}_{\mathsf{ID_A}} \cdot \ \mathbf{e} \mod p$). The TA gives ω_{A} to user A as her private key. For B and all other users, the TA generates their private keys in the same way.

User B, who wants to securely communicate with A, can now compute A's "public key" y_{A} $(= \alpha^{\omega_{\mathsf{A}}} \mod q)$ by himself. First, B computes $\mathbf{s}_{\mathsf{ID_A}} \in \{0,1\}^z \ \leftarrow \ \mathrm{hid}(\mathsf{ID_A})$. Then, he computes A's "public key": $y'_{\mathsf{A}} \in G_p \ \leftarrow \ \prod_{i=1}^{z} (\mathbf{f}[i])^{\mathbf{s}_{\mathsf{ID_A}}[i]} \mod q$. Note that the exponentiation with $\mathbf{s}_{\mathsf{ID_A}}[i]$ (which can only have values 0 or 1) is practically just a (computationally almost gratis) selection of *actual* factors for the *multiplication* from a set of z *potential* factors, namely from $\mathbf{f}$. When looking at how the elements of $\mathbf{f}$ were computed from α and $\mathbf{e}$, we see that $y'_{\mathsf{A}} \ = \ \prod_{i=1}^{z} (\mathbf{f}[i])^{\mathbf{s}_{\mathsf{ID_A}}[i]} \mod q \ = \ \prod_{i=1}^{z} \left(\alpha^{\mathbf{e}[i]}\right)^{\mathbf{s}_{\mathsf{ID_A}}[i]} \mod q \ = \ \alpha^{\sum_{i=1}^{z}\left(\mathbf{s}_{\mathsf{ID_A}}[i] \ \cdot \ \mathbf{e}[i]\right)} \mod q \ = \ \alpha^{\omega_{\mathsf{A}}} \mod q \ = \ y_{\mathsf{A}}$. Finally, B computes the key shared with A as $(y'_{\mathsf{A}})^{\omega_{\mathsf{B}}} \mod q$ $(= \alpha^{\sum_{i=1}^{z}\left(\mathbf{s}_{\mathsf{ID_A}}[i] \ \cdot \ \mathbf{e}[i]\right) \ \cdot \ \sum_{i=1}^{z}\left(\mathbf{s}_{\mathsf{ID_B}}[i] \ \cdot \ \mathbf{e}[i]\right)} \mod q)$. Likewise, also A first computes B's "public key" y_{B} $(= \alpha^{\omega_{\mathsf{B}}} \mod q)$, and then the key shared with B as $(y'_{\mathsf{B}})^{\omega_{\mathsf{A}}} \mod q$ $(= \alpha^{\sum_{i=1}^{z}\left(\mathbf{s}_{\mathsf{ID_B}}[i] \ \cdot \ \mathbf{e}[i]\right) \ \cdot \ \sum_{i=1}^{z}\left(\mathbf{s}_{\mathsf{ID_A}}[i] \ \cdot \ \mathbf{e}[i]\right)} \mod q)$.

The mechanism described above shows a way to computation-efficient non-interactive key distribution, but it is not yet a solution. The protocol is secure against attacks by colluding users (equipped with private keys) only as long as the vectors $\mathbf{s}_{\mathsf{ID_A}}$ and $\mathbf{s}_{\mathsf{ID_B}}$ are linearly independent from the vectors $\mathbf{s}_{\mathsf{ID}_j}$, i.e., the hashes of the identities ID_j, of all users with private keys in the system. If they are not, then the private key $\omega_{\mathsf{ID_A}}$ of a user A with the identity $\mathsf{ID_A}$ whose hash $\mathbf{s}_{\mathsf{ID_A}}$ is a linear combination of the hashes $\mathbf{s}_{\mathsf{ID}_1}, ..., \mathbf{s}_{\mathsf{ID}_n}$ of the identities $\mathsf{ID}_1, ..., \mathsf{ID}_n$ of a set of n other users can be computed by these users if they collude in an attack on the user A. First, using Gaussian elimination, the colluders determine a set of coefficients $c_j \in \mathbb{Z}_p$ that solves the vector congruence $\mathbf{s}_{\mathsf{ID_A}} \equiv \sum_{j=1}^{n} c_j \mathbf{s}_{\mathsf{ID}_j} \pmod p$, and then compute $\omega_{\mathsf{ID_A}} \ \leftarrow \ \sum_{j=1}^{n} \ c_j \omega_{\mathsf{ID}_j} \mod p$. This works analogously if $\mathbf{s}_{\mathsf{ID_B}}$ is not linearly independent. In order to get to a protocol that is secure also if linear independence is not guaranteed, two further steps are necessary.

Imagine the following: In order to reduce the computational complexity of the shared-key generation (the runtime computation cost), user A precomputes for herself a vector $\mathbf{f}_{\mathsf{ID_A}}$ $(\forall i \in \{1, ..., z\}: \ \mathbf{f}_{\mathsf{ID_A}}[i] \in G_p \ \leftarrow \ \mathbf{f}[i]^{\omega_{\mathsf{A}}} \mod q)$ such that when A later wants to compute a key shared with another user (say, B), A can compute the shared key more efficiently (without an exponentiation needed) just as $\prod_{i=1}^{z} \mathbf{f}_{\mathsf{ID_A}}^{\mathbf{s}_{\mathsf{ID_B}}[i]} \mod q$ (where $\mathbf{s}_{\mathsf{ID_B}}$ is computed from $\mathsf{ID_B}$ as described above).

Now imagine that A has *not herself* computed $\mathbf{f}_{\mathsf{ID_A}}$ but has received it from the TA as her private key, instead of ω_{A}. Further imagine the same is done for all users, the private key of a user (with identity ID_j) is now generally a vector $\mathbf{f}_{\mathsf{ID}_j}$ where $\forall i \in \{1, ..., z\}: \ \mathbf{f}_{\mathsf{ID}_j}[i] \in G_p \ = \ \mathbf{f}[i]^{\omega_{\mathsf{ID}_j}} \mod q \ = \ \alpha^{\left(\mathbf{e}[i] \ \cdot \ \omega_{\mathsf{ID}_j}\right)} \mod q \ = \ \alpha^{\left(\mathbf{e}[i] \ \cdot \ \sum_{k=1}^{z}\left(\mathbf{s}_{\mathsf{ID}_j}[k] \ \cdot \ \mathbf{e}[k]\right)\right)} \mod q$.

Also this protocol is not yet secure against colluding users in cases where the $\mathbf{s}_{\mathsf{ID}_j}$'s of the users are *not* linearly independent. In such cases, the private key $\mathbf{f}_{\mathsf{ID_A}}$ of a user A with the identity $\mathsf{ID_A}$ whose hash $\mathbf{s}_{\mathsf{ID_A}}$ is a linear combination of the hashes $\mathbf{s}_{\mathsf{ID}_1}, ..., \mathbf{s}_{\mathsf{ID}_n}$ of the identities $\mathsf{ID}_1, ..., \mathsf{ID}_n$ of a set of n other users can still be computed by these users if they collude. First, using Gaussian elimination, the colluders determine a set of coefficients $c_j \in \mathbb{Z}_p$ that solves the vector congruence $\mathbf{s}_{\mathsf{ID_A}} \equiv \sum_{j=1}^{n} c_j \mathbf{s}_{\mathsf{ID}_j} \pmod p$, and then compute $\mathbf{f}_{\mathsf{ID_A}}$: $\forall i \in \{1, ..., z\}: \ \mathbf{f}_{\mathsf{ID_A}}[i] \ \leftarrow \ \prod_{j=1}^{n} \ \left(\mathbf{f}_{\mathsf{ID}_j}[i]\right)^{c_j} \mod q$.

In order to achieve collusion resistance, the $\mathbf{f}_{\mathsf{ID}_j}$'s above should be given to the users only user-specifically blinded. As a means to achieve this, we first introduce an "additional level". Let G_q be an algebraic group of order q and let β be a generator of G_q. Note that the order of G_q is identical with the modulus in G_p. Shared keys are now computed on both sides by taking β to the power (in G_q) of what previously was the shared key.

For instance, A computes the key shared with B as $\beta^{\left(\prod_{i=1}^{z} \mathbf{f}_{\mathsf{ID_A}}^{\mathsf{s}_{\mathsf{ID_B}}[i]} \mod q\right)}$, and B computes the key shared with A as $\beta^{\left(\prod_{i=1}^{z} \mathbf{f}_{\mathsf{ID_B}}^{\mathsf{s}_{\mathsf{ID_A}}[i]} \mod q\right)}$.

Using this "two levels" construction, the $\mathbf{f}_{\mathsf{ID}_j}$'s given to the users can now be *user-specifically blinded copies* of what they were before. As its private key, A now gets from the TA a vector $\mathbf{f}_{\mathsf{ID_A}} \in G_p^z$ where the elements were computed: $\forall i \in \{1, ..., z\} : \ \mathbf{f}_{\mathsf{ID_A}}[i] \in G_p \ \leftarrow \ \alpha^{\left(\mathbf{e}[i] \sum_{k=1}^{z}\left(\mathsf{s}_{\mathsf{ID_A}}[k] \cdot \mathbf{e}[k]\right) + r_{\mathsf{ID_A}} \mod p\right)} \mod q$ where $r_{\mathsf{ID_A}}$ was randomly chosen from $\mathbb{Z}_p$, specifically for $\mathsf{ID_A}$. The addition of $r_{\mathsf{ID_A}}$ in the exponent is for the elements of $\mathbf{f}_{\mathsf{ID_A}}$ effectively a *multiplicative blinding* (of what the elements were in the previous protocol) with $\alpha^{r_{\mathsf{ID_A}}} \mod q$, in effect with a random element of G_p. As a means for the "neutralization" of the blinding of the elements of the private key $\mathbf{f}_{\mathsf{ID_A}}$ when using them for the computation of a shared key, A gets a *key-enabler* that corresponds to her private key. A's *personal key* consists now of two parts, her *private key* $\mathbf{f}_{\mathsf{ID_A}}$ and the corresponding *key-enabler* $\mathbf{b}_{\mathsf{ID_A}}$, a vector of $z+1$ elements in G_q (the index here running from 0 to z). The elements of $\mathbf{b}_{\mathsf{ID_A}}$ are computed: $\forall h \in \{0, ..., z\} : \ \mathbf{b}_{\mathsf{ID_A}}[h] \in G_q \ \leftarrow \ \beta^{\left(\alpha^{\left(-h \cdot r_{\mathsf{ID_A}} \mod p\right)} \mod q\right)}$. Note that, in contrast to the private key $\mathbf{f}_{\mathsf{ID_A}}$, the key-enabler $\mathbf{b}_{\mathsf{ID_A}}$ need not be kept secret.

Using her personal key (i.e., private key and key-enabler), A computes the key shared with B now as follows. First, A computes $\mathsf{s}_{\mathsf{ID_B}} \leftarrow \mathrm{hid}(\mathsf{ID_B})$ and determines $\mathsf{s}_{\mathsf{ID_B}}$'s Hamming weight $h_{\mathsf{ID_B}}$ by counting the ones in $\mathsf{s}_{\mathsf{ID_B}}$. Then, A computes the shared key as $(\mathbf{b}_{\mathsf{ID_A}}[h_{\mathsf{ID_B}}])^{\left(\prod_{i=1}^{z} \mathbf{f}_{\mathsf{ID_A}}^{\mathsf{s}_{\mathsf{ID_B}}[i]} \mod q\right)}$. Likewise, B computes the key shared with A as $(\mathbf{b}_{\mathsf{ID_B}}[h_{\mathsf{ID_A}}])^{\left(\prod_{i=1}^{z} \mathbf{f}_{\mathsf{ID_B}}^{\mathsf{s}_{\mathsf{ID_A}}[i]} \mod q\right)}$. Looking at A's shared-key computation, note that for any $h_{\mathsf{ID_B}}$ the base is $\beta^{\left(\alpha^{-\left(h_{\mathsf{ID_B}} \cdot r_{\mathsf{ID_A}}\right)} \mod q\right)}$ and the exponent is $\alpha^{\left(h_{\mathsf{ID_B}} \cdot r_{\mathsf{ID_A}}\right)} \cdot \alpha^{\left(\sum_{i=1}^{z}\left(\mathsf{s}_{\mathsf{ID_A}}[i] \cdot \mathbf{e}[i]\right) \cdot \sum_{i=1}^{z}\left(\mathsf{s}_{\mathsf{ID_B}}[i] \cdot \mathbf{e}[i]\right)\right)} \mod q$. Evidently, the two $\alpha^{\left(h_{\mathsf{ID_B}} \cdot r_{\mathsf{ID_A}}\right)}$'s cancel each other out, i.e., the blinding is neutralized, the result of A's shared-key computation is $\beta^{\left(\alpha^{\left(\sum_{i=1}^{z}\left(\mathsf{s}_{\mathsf{ID_A}}[i] \cdot \mathbf{e}[i]\right) \cdot \sum_{i=1}^{z}\left(\mathsf{s}_{\mathsf{ID_B}}[i] \cdot \mathbf{e}[i]\right)\right)} \mod q\right)}$. When B computes the key shared with A, the blinding and its neutralization on his side work exactly like this. Thus, B gets to the same result.

A's personal key is now secure against attacks by colluding users with own personal keys even if $\mathsf{s}_{\mathsf{ID_A}}$ is a linear combination of the $\mathsf{s}_{\mathsf{ID}_j}$'s resulting from the identities of the colluders. The colluders can still determine the coefficients $c_j \in \mathbb{Z}_p$ that solve the congruence $\mathsf{s}_{\mathsf{ID_A}} \equiv \sum_{j=1}^{n} c_j \mathsf{s}_{\mathsf{ID}_j} \pmod{p}$, but this does not help them anymore because the elements of each $\mathbf{f}_{\mathsf{ID}_j}$ are now products, each one including an undisclosed ID_j-specific blinding factor.

Note that for the shared-key generation in the last protocol, the users do not need the base α or the (former) system parameter $\mathbf{f}$. Now, the TA keeps them secret. The secrecy of $\mathbf{f}$ and α means a big difference concerning the requirements on the choice of the group G_p, i.e., of q and p. As long as $\mathbf{f}$ and α were public, the foundational problem (which must have an average complexity $\geq 2^k$ in G_p) was a descendant of the Diffie-Hellman problem (DHP, see [10], or, e.g., [20] or [22]), thus not harder than DLP. The problem in G_p we have now looks at the surface like a descendant of DHP and DLP, but is in fact fundamentally different. The secrecy of the base α together with the multiplication of what were the elements of $\mathbf{f}$ in the previous problem with an unknown identity-specific factor (namely, $\alpha^{r_{\mathsf{ID_A}}} \mod q$ for $\mathsf{ID_A}$) means that algorithms for DLP, including the adaptation of the Number Field Sieve for DLP [30] and Pollard's *rho* algorithm for logarithms [28], do not solve the problem in G_p. As a very welcome consequence, a bit length of about $2k$ is now sufficient for q.[4] Concerning the computational complexity of the shared-key generation, this means that the (fixed-based) exponentiation in G_q does not cost more than what one is used to (for a fixed-base exponentiation) from conventional (i.e., not pairing-based) elliptic-curve based public cryptography (see, e.g., [15] or [7]), and the multiplications in G_p become inexpensive.

In the schemes so far, the users get their personal keys from a single TA. This means that the TA's operations, and particularly its master key, are a single point of vulnerability, and that the users have no protection against dishonest behavior of the TA. In a variation of the last protocol, we now introduce a multitude of $t \geq 1$ (independent) authorities $\mathsf{TA}_1, ..., \mathsf{TA}_t$ into the system. G_p, G_q, β, and z remain the same for the whole system, but each TA_x has its own master key, consisting of $\alpha_{\mathsf{TA}_x} \in G_p$ and $\mathbf{e}_{\mathsf{TA}_x} \in \mathbb{Z}_p^z$. The effect of the variation is that no single TA_x in the system (and also nobody else who has learned a TA_x's master key) can learn the shared key of a pair of users, with the only exception that *all* TA_xs collaborate, or somebody has learned *all* master keys. Instead of getting $\mathbf{f}_{\mathsf{ID_A}}$ from a single TA, A now gets from each TA_x in $\{\mathsf{TA}_1, ..., \mathsf{TA}_t\}$ an $\mathbf{f}_{[\mathsf{ID_A}, \mathsf{TA}_x]}$ which has (not different from what the TA's does in the previous protocol) been computed by the TA_x using its master

[4] The situation in G_p bears some similarity with discrete-logarithm based digital signature schemes, for instance, with $\mathbb{Z}_p^*$ in DSA [24] or Schnorr signatures [31], and with $\mathbb{Z}_n^*$ in ECDSA [24]. Although most operations happen in these groups, it is not required that DLP is infeasible there. Infeasibility of DLP is required only for the underlying group where p (DSA, Schnorr) resp. n (ECDSA) is the *order* .

key $\langle \alpha_{\mathsf{TA}_x}, \mathbf{e}_{\mathsf{TA}_x} \rangle$ as follows: $\forall i \in \{1, ..., z\}$: $\mathbf{f}_{[\mathsf{ID_A}, \mathsf{TA}_x]}[i] \in G_p \;\leftarrow\; \alpha_{\mathsf{TA}_x}^{\left(\mathbf{e}_{\mathsf{TA}_x}[i] \sum_{k=1}^{z} \left(\mathbf{s}_{\mathsf{ID_A}}[k] \,\cdot\, \mathbf{e}_{\mathsf{TA}_x}[k]\right) \,+\, r_{[\mathsf{ID_A}, \mathsf{TA}_x]} \mod p\right)} \mod q$ where the TA_x has $r_{[\mathsf{ID_A}, \mathsf{TA}_x]}$ randomly chosen from $\mathbb{Z}_p$, specifically for $\mathsf{ID_A}$. From these $\mathbf{f}_{[\mathsf{ID_A}, \mathsf{TA}_x]}$'s, A computes $\mathbf{f}_{\mathsf{ID_A}}$ herself: $\forall i \in \{1, ..., z\}$: $\mathbf{f}_{\mathsf{ID_A}}[i] \in G_p \;\leftarrow\; \prod_{x=1}^{t} \mathbf{f}_{[\mathsf{ID_A}, \mathsf{TA}_x]}[i] \mod q$. A creates also the corresponding key-enabler $\mathbf{b}_{\mathsf{ID_A}}$ and assigns β to all its $z + 1$ elements as the *initial* value. Then, each time A requests from a TA_x the vector $\mathbf{f}_{[\mathsf{ID_A}, \mathsf{TA}_x]}$, it sends along $\mathbf{b}_{\mathsf{ID_A}}$ in its current state. The TA_x does not only compute $\mathbf{f}_{[\mathsf{ID_A}, \mathsf{TA}_x]}$ but also updates $\mathbf{b}_{\mathsf{ID_A}}$ accordingly, i.e., according to α_{TA_x} and $r_{[\mathsf{ID_A}, \mathsf{TA}_x]}$, as follows: $\forall h \in \{0, ..., z\}$: $\mathbf{b}_{\mathsf{ID_A}}[h] \in G_q \;\leftarrow\; \left(\mathbf{b}_{\mathsf{ID_A}}[h]\right)^{\left(\alpha_{\mathsf{TA}_x}^{\left(-h \,\cdot\, r_{[\mathsf{ID_A}, \mathsf{TA}_x]} \mod p\right)} \mod q\right)}$. Finally, after having been updated by each TA_x in $\{\mathsf{TA}_1, ..., \mathsf{TA}_t\}$, $\mathbf{b}_{\mathsf{ID_A}}$ has this value: $\forall h \in \{0, ..., z\}$: $\mathbf{b}_{\mathsf{ID_A}}[h] = \beta^{\left(\prod_{x=1}^{t} \alpha_{\mathsf{TA}_x}^{\left(-h \,\cdot\, r_{[\mathsf{ID_A}, \mathsf{TA}_x]} \mod p\right)} \mod q\right)}$. A computes the key shared with B exactly as in the last protocol with a single TA, and so does B. Looking at A's shared-key computation, note that for any $h_{\mathsf{ID_B}}$ the base is $\beta^{\left(\prod_{x=1}^{t} \alpha_{\mathsf{TA}_x}^{-\left(h_{\mathsf{ID_B}} \,\cdot\, r_{[\mathsf{ID_A}, \mathsf{TA}_x]} \mod p\right)} \mod q\right)}$ and the exponent is $\prod_{x=1}^{t} \alpha_{\mathsf{TA}_x}^{\left(h_{\mathsf{ID_B}} \,\cdot\, r_{[\mathsf{ID_A}, \mathsf{TA}_x]} \mod p\right)} \cdot \prod_{x=1}^{t} \alpha_{\mathsf{TA}_x}^{\left(\sum_{i=1}^{z} \left(\mathbf{s}_{\mathsf{ID_A}}[i] \,\cdot\, \mathbf{e}_{\mathsf{TA}_x}[i]\right) \,\cdot\, \sum_{i=1}^{z} \left(\mathbf{s}_{\mathsf{ID_B}}[i] \,\cdot\, \mathbf{e}_{\mathsf{TA}_x}[i]\right)\right)} \mod q$. Evidently, the two $\left(\prod_{x=1}^{t} \alpha_{\mathsf{TA}_x}^{\left(h_{\mathsf{ID_B}} \,\cdot\, r_{[\mathsf{ID_A}, \mathsf{TA}_x]} \mod p\right)}\right)$'s cancel each other out, i.e., the blinding is neutralized, the result of A's shared-key computation is $\beta^{\left(\prod_{x=1}^{t} \alpha_{\mathsf{TA}_x}^{\left(\sum_{i=1}^{z} \left(\mathbf{s}_{\mathsf{ID_A}}[i] \,\cdot\, \mathbf{e}_{\mathsf{TA}_x}[i]\right) \,\cdot\, \sum_{i=1}^{z} \left(\mathbf{s}_{\mathsf{ID_B}}[i] \,\cdot\, \mathbf{e}_{\mathsf{TA}_x}[i]\right)\right)} \mod q\right)}$. When B computes the key shared with A, the blinding and its neutralization on his side work exactly like this. Thus, B gets to the same result. Note that this result cannot be computed by anybody else than A and B who does not know the master keys $\langle \alpha_{\mathsf{TA}_x}, \mathbf{e}_{\mathsf{TA}_x} \rangle$ of *all* TAs.

A user's personal key (private key and key-enabler) in the protocol above is significantly larger than what one is used to from most public-key schemes, including IBC. A final (optional) refinement halves the size of the personal key. Now, let *hid* not be a common cryptographic hash function with outputs of an unpredictable Hamming weight between 0 and z. Instead, let *hid* be a *special* cryptographic hash function with a *prescribed* $h \in \{1, ..., z - 1\}$ where all outputs of *hid* have the *same* Hamming weight h.[5] Then, user A's key-enabler need not be a whole vector of z+1 "deblinding bases", one for each possible Hamming weight of the $\mathbf{s}_{\mathsf{ID}_j}$ resulting from the communication partner's identity ID_j. Instead of the *vector* $\mathbf{b}_{\mathsf{ID_A}}$, A's key-enabler is now just *one scalar* "deblinding base" $b_{\mathsf{ID_A}}$ which all the TA_x's in $\{\mathsf{TA}_1, ..., \mathsf{TA}_t\}$ have, one after the other, starting with the initial value β, processed as above, until it has its final value $\beta^{\left(\prod_{x=1}^{t} \alpha_{\mathsf{TA}_x}^{\left(-h \,\cdot\, r_{[\mathsf{ID_A}, \mathsf{TA}_x]} \mod p\right)} \mod q\right)}$. Now, A computes, using her private key $\mathbf{f}_{\mathsf{ID_A}}$ and key-enabler $b_{\mathsf{ID_A}}$, the key shared with B as follows. First, A computes $\mathbf{s}_{\mathsf{ID_B}} \leftarrow \text{hid}(\mathsf{ID_B})$. Then, A computes the shared key as $b_{\mathsf{ID_A}}^{\left(\prod_{i=1}^{z} \mathbf{f}_{\mathsf{ID_A}}^{\mathbf{s}_{\mathsf{ID_B}}[i]} \mod q\right)}$. Likewise, B computes the key shared with A as $b_{\mathsf{ID_B}}^{\left(\prod_{i=1}^{z} \mathbf{f}_{\mathsf{ID_B}}^{\mathbf{s}_{\mathsf{ID_A}}[i]} \mod q\right)}$. Looking at A's shared-key computation, note that the base is $\beta^{\left(\prod_{x=1}^{t} \alpha_{\mathsf{TA}_x}^{-\left(h \,\cdot\, r_{[\mathsf{ID_A}, \mathsf{TA}_x]} \mod p\right)} \mod q\right)}$ and the exponent is $\prod_{x=1}^{t} \alpha_{\mathsf{TA}_x}^{\left(h \,\cdot\, r_{[\mathsf{ID_A}, \mathsf{TA}_x]} \mod p\right)} \cdot \prod_{x=1}^{t} \alpha_{\mathsf{TA}_x}^{\left(\sum_{i=1}^{z} \left(\mathbf{s}_{\mathsf{ID_A}}[i] \,\cdot\, \mathbf{e}_{\mathsf{TA}_x}[i]\right) \,\cdot\, \sum_{i=1}^{z} \left(\mathbf{s}_{\mathsf{ID_B}}[i] \,\cdot\, \mathbf{e}_{\mathsf{TA}_x}[i]\right)\right)} \mod q$. Evidently, the two $\left(\prod_{x=1}^{t} \alpha_{\mathsf{TA}_x}^{\left(h \,\cdot\, r_{[\mathsf{ID_A}, \mathsf{TA}_x]} \mod p\right)}\right)$'s cancel each other out, i.e., the blinding is neutralized, the result of A's shared-key computation is $\beta^{\left(\prod_{x=1}^{t} \alpha_{\mathsf{TA}_x}^{\left(\sum_{i=1}^{z} \left(\mathbf{s}_{\mathsf{ID_A}}[i] \,\cdot\, \mathbf{e}_{\mathsf{TA}_x}[i]\right) \,\cdot\, \sum_{i=1}^{z} \left(\mathbf{s}_{\mathsf{ID_B}}[i] \,\cdot\, \mathbf{e}_{\mathsf{TA}_x}[i]\right)\right)} \mod q\right)}$. When B computes the key shared with A, the blinding and its neutralization on his side work exactly like this. Thus, B gets to the same result. Note that also this result cannot be computed by anybody else than A and B who does not know the master keys $\langle \alpha_{\mathsf{TA}_x}, \mathbf{e}_{\mathsf{TA}_x} \rangle$ of *all* TAs.

Avoiding the need to deal with $z + 1$ "deblinding bases" (instead of just one) appears to be sensible especially if one considers that the exponentiations of these bases are *fixed-base* exponentiations. One should expect that implementations of the protocol will use a runtime-optimized algorithm for fixed-base exponentiation (for an overview, see, e.g., [22]) that does some precomputations. Without the last refinement, not only is a user's key material about double the size (if G_q is an elliptic-curve group), but more importantly, the amount of precomputed data to be stored for faster exponentiations in G_q later is z times higher.

The protocol above is the core of our NIKD-DPKG scheme, which we will present in detail in Section 4. Before, we provide a more general introduction and definitions of NIKD-DPKG schemes and their security.

[5] Two examples how such a function can be constructed from a standard cryptographic hash function, such as SHA [25], are given in Appendix B.

3 Non-interactive Key Distribution with Distributed Private-key Generation

A scheme for the *non-interactive key distribution with distributed private-key generation* (NIKD-DPKG) enables pairs of users with unique identities (practically, bit strings) to generate a pairwise-shared key, each one of the two users by himself, from his personal (long-term) key, the identity of the other user, and the public system parameters. Nobody else than these two users can learn their shared key, with the only exception that somebody has learned the master keys of all authorities, or that all authorities cooperate in order to compute this shared key. An NIKD-DPKG scheme consists of five corresponding algorithms.

Initially, the algorithm $\mathsf{SetupSystem}$ is performed, by some standardization body, or together by the key-issuing authorities in the system. It sets the system parameters. They are public. Then, by running algorithm $\mathsf{SetupKEC}$, each authority sets up its *Key-Enabling Center* (KEC), the entity that acts for the authority and serves the users when they set up their personal keys. The setup of a KEC includes the generation of the KEC's master key. Each user by himself sets up his personal key, i.e., the long-term key that corresponds to his identity, by running algorithm $\mathsf{SetupPersonalKey}$. The user's personal key consists of the user's private key and a corresponding (non-private) key-enabler. The algorithm $\mathsf{SetupPersonalKey}$, running for the user on the user's side, contacts, one after the other, the KECs in the system and requests each of them to enable the personal key being set up for the user (by contributing the KEC's share to the user's private key and processing the user's key-enabler accordingly). When requested by a user (precisely by the algorithm $\mathsf{SetupPersonalKey}$ run by this user on his side) to enable the user's personal key, the KEC runs algorithm $\mathsf{EnablePersonalKey}$, with the user's identity and the user's key-enabler (in its current state) given as input. The KEC returns the algorithm's results, i.e., the KEC's contribution to the user's private key and the user's correspondingly updated key-enabler, to the algorithm $\mathsf{SetupPersonalKey}$ running on the user's side.[6] Later, when two users, after having set up their personal keys, want to establish a shared key, each of the two (independently) runs algorithm $\mathsf{SharedKey}$, each party giving as the input its own personal key (private key and key-enabler) and the other party's identity. In detail, the inputs and outputs of the five algorithms are: [7]

SetupSystem: takes the security parameter 1^k and returns the public system parameters params. The params include a list of the KECs in the system (and their contact information). The params are made public.

SetupKEC: takes the public parameters params and returns the KEC's master key keckey, which shall be known only to this KEC (and the authority operating it).

SetupPersonalKey: takes the public parameters params and an identity $\mathsf{ID} \in \{0,1\}^*$. After interaction with all KECs (which on their side run algorithm $\mathsf{EnablePersonalKey}$ in order to contribute to the personal key being set up), the algorithm returns the personal key κ_{ID}, which consists of the private key f_{ID} from some private-key space $\mathcal{F}$ and the corresponding (non-private) key-enabler b_{ID} from some key-enabler space $\mathcal{B}$.

EnablePersonalKey: takes the public parameters params, the executor's master key keckey, an arbitrary identity $\mathsf{ID} \in \{0,1\}^*$, and the corresponding key enabler $b_{\mathsf{ID}} \in \mathcal{B}$. The algorithm returns a private-key share v_{ID} from some private-key-share space $\mathcal{V}$ and the accordingly processed b_{ID}.

SharedKey: takes the public parameters params, the personal key $\kappa_{\mathsf{ID_A}}$ of the executor, and another identity $\mathsf{ID_B} \in \{0,1\}^*$. It returns the key $\gamma_{\mathsf{A,B}}$ shared by the two parties with the identities $\mathsf{ID_A}$ and $\mathsf{ID_B}$. $\gamma_{\mathsf{A,B}}$ is from some shared-key space $\mathcal{K}$.

Consistency. The five algorithms of an NIKD-DPKG scheme must satisfy the following consistency constraint: When $\kappa_{\mathsf{ID_A}}$ is the personal key generated by algorithm $\mathsf{SetupPersonalKey}$ when it is given $\mathsf{ID_A}$ as input, and $\kappa_{\mathsf{ID_B}}$ is the personal key generated by algorithm $\mathsf{SetupPersonalKey}$ when given $\mathsf{ID_B}$, then $\mathsf{SharedKey}(\mathsf{params}, \kappa_{\mathsf{ID_A}}, \mathsf{ID_B}) = \mathsf{SharedKey}(\mathsf{params}, \kappa_{\mathsf{ID_B}}, \mathsf{ID_A})$. This ensures that users A and B can indeed generate a shared key without any interaction. We will normally assume that $\mathcal{K}$, the space of shared keys, is $\{0,1\}^{l(k)}$ for some function $l(k) \geq k$. In practice, this can be arranged by hashing a "raw" key.

Security. Intuitively, an NIKD-DPKG scheme Φ is secure if for no pair of identities anyone else than the two users with these identities, including the authorities and their KECs, can in any way learn the two users' shared key. The only exception is for *lawful* interception, that is, if *all* authorities (i.e., their KECs) work together, then they shall be able to reproduce the key shared by any two users (as may be ordered by a law court, e.g., for criminal investigations). Our attack scenario includes that an attacker may have learned other shared keys, just not the one that he wants to attack. As usual for identity-based cryptography, the attack scenario further includes

[6] As usual for identity-based cryptography, before contributing to the generation of the user's personal key, the KEC authenticates the user in some external way. The privacy and the authenticity of the information passed between the two is secured by external means.

[7] Unfortunately, the generality of our definitions of NIKD-DPKG and its security is somewhat limited, they obviously fit only schemes with a certain pattern of interaction for establishing the users' personal keys.

that a group of registered users, equipped with own personal keys, may collude. Even such groups of colluding users shall not be able to derive the secret shared by any other two users. Our attack scenario also includes that some (but never all) KECs may be corrupted, or that some dishonest authorities (but never all) may participate in a collusion attack. Formally, our security model is stated in terms of a game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. The game models the most powerful attack imaginable: a collusion of all KECs except one integer (honest and not corrupted) KEC and of all registered users except two targets of the attack. They can freely be chosen by the attackers, at any time, and adaptively. No attack where not *all* KECs keys *leak bits* of their master keys are less powerful. Thus, all other attacks are covered by this model. The game is:

System Setup: The challenger $\mathcal{C}$ takes the security parameter 1^k and the number of authorities t (each authority assumed to operate one KEC). First, $\mathcal{C}$ runs the $\mathsf{SetupSystem}$ algorithm. $\mathcal{C}$ gives the adversary $\mathcal{A}$ the params.

KEC Setup: $\mathcal{A}$ selects from the ordered list of Key-Enabling Center identities $\{\mathsf{KEC}_1, ..., \mathsf{KEC}_t\}$ the one (integer) Key-Enabling Center IKEC that shall neither be corrupted nor run by a dishonest authority. $\mathcal{C}$ sets up the Key-Enabling Center IKEC by running the algorithm $\mathsf{SetupKEC}$. It keeps IKEC's resulting master key $\mathsf{keckey}_{\mathsf{IKEC}}$ to itself. $\mathcal{A}$ sets up the other Key-Enabling Centers $\{\mathsf{KEC}_1, ..., \mathsf{KEC}_t\} \backslash \mathsf{IKEC}$ by running the algorithm $\mathsf{SetupKEC}$.

Queries Phase: $\mathcal{A}$ issues one after the other, queries q_j where each query is one of:

– SetupPersonalKey query ($\mathsf{ID}_j \in \{0,1\}^*$): $\mathcal{C}$ responds by running algorithm $\mathsf{SetupPersonalKey}$ with input argument ID_j.

For each of the Key-Enabling Center identities $\{\mathsf{KEC}_1, ..., \mathsf{KEC}_t\} \backslash \mathsf{IKEC}$, the algorithm $\mathsf{SetupPersonalKey}$ interacts with $\mathcal{A}$ (which impersonates the KECs that participate in the collusion attack, i.e., all KECs except IKEC), from which, handing in the identity ID_j and its key-enabler b_{ID_j} in its current state, algorithm $\mathsf{SetupPersonalKey}$ requests to run algorithm $\mathsf{EnablePersonalKey}$ in order to compute the respective KEC_x's contribution $v_{\mathsf{ID}_j,\mathsf{KEC}_x}$ to ID_j's private key f_{ID_j} and the corresponding update of b_{ID_j}. Algorithm $\mathsf{SetupPersonalKey}$ uses the $v_{\mathsf{ID}_j,\mathsf{KEC}_x}$ from $\mathcal{A}$'s response to compute ID_j's private key f_{ID_j} and maintains b_{ID_j} as ID_j's key-enabler.

The integer Key-Enabling Center (IKEC) is impersonated by $\mathcal{C}$ itself: $\mathcal{C}$ runs algorithm $\mathsf{EnablePersonalKey}$ with $\mathsf{ID}_j, b_{\mathsf{ID}_j}$ in its current state and IKEC's master key $\mathsf{keckey}_{\mathsf{IKEC}}$ given as the algorithm's input. Algorithm $\mathsf{SetupPersonalKey}$ uses the resulting $v_{\mathsf{ID}_j,\mathsf{IKEC}}$ to compute ID_j's private key f_{ID_j} and maintains b_{ID_j} as ID_j's key-enabler.

The temporal order of algorithm $\mathsf{SetupPersonalKey}$'s requests (to perform algorithm $\mathsf{EnablePersonalKey}$ for the single KEC identities) to $\mathcal{A}$ and to $\mathcal{C}$ itself is the same as the order of $\{\mathsf{KEC}_1, ..., \mathsf{KEC}_t\}$.

$\mathcal{C}$ keeps ID_j's resulting personal key $\kappa_{\mathsf{ID}_j} = \langle f_{\mathsf{ID}_j}, b_{\mathsf{ID}_j}\rangle$ to itself. $\mathcal{C}$ adds the pair $\langle \mathsf{ID}_j, \kappa_{\mathsf{ID}_j}\rangle$ to the list L of personal keys already issued (if an entry for ID_j already exists, it is replaced).

– EnablePersonalKey query ($\mathsf{ID}_j \in \{0,1\}^*, b_{\mathsf{ID}_j} \in \mathcal{B}$): $\mathcal{C}$ responds by running algorithm $\mathsf{EnablePersonalKey}$ with $\mathsf{ID}_j, b_{\mathsf{ID}_j}$, and IKEC's master key $\mathsf{keckey}_{\mathsf{IKEC}}$ given as the input. $\mathcal{C}$ sends the resulting $v_{\mathsf{ID}_j,\mathsf{IKEC}} \in \mathcal{V}$ and the accordingly updated b_{ID_j} to $\mathcal{A}$. [8]

– RevealSharedKey query ($\mathsf{ID_A} \in \{0,1\}^*, \mathsf{ID_B} \in \{0,1\}^*$: $\mathcal{C}$ first retrieves $\mathsf{ID_A}$'s personal key $\kappa_{\mathsf{ID_A}}$ from L. If it is not found in L, then $\mathcal{C}$ sets up $\kappa_{\mathsf{ID_A}}$, by doing exactly the same as when responding to a $\mathsf{SetupPersonalKey}$ query for $\mathsf{ID_A}$ (see above), except that it does not give $\mathcal{A}$ the resulting $\kappa_{\mathsf{ID_A}}$. Instead, $\mathcal{C}$ runs algorithm $\mathsf{SharedKey}$ with the input arguments $\kappa_{\mathsf{ID_A}}$ and $\mathsf{ID_B}$ and gives $\mathcal{A}$ the result of algorithm $\mathsf{SharedKey}$.

The queries may be asked adaptively, that is, each query q_j may depend on the previous replies.

For the final test, $\mathcal{A}$ selects a pair of identities $\mathsf{ID_A}, \mathsf{ID_B} \in \{0,1\}^*$ with which no $\mathsf{RevealSharedKey}$ query had been issued, where $\mathsf{ID_A} \neq \mathsf{ID_B}$, and where no $\mathsf{EnablePersonalKey}$ query with any of the two identities $\mathsf{ID_A}, \mathsf{ID_B}$ had been issued. $\mathcal{A}$ gives $\mathsf{ID_A}, \mathsf{ID_B}$ to $\mathcal{C}$. $\mathcal{C}$ first computes $\mathsf{ID_A}$ and $\mathsf{ID_B}$'s shared key $\gamma_{\mathsf{A,B}}$ in the same way it has computed shared keys in order to respond to $\mathsf{RevealSharedKey}$ queries (see above). $\mathcal{C}$ then randomly selects b from $\{0,1\}$. If $b = 0$ then $\mathcal{C}$ chooses $\gamma_{\mathsf{A,B}}$ as the value for $\xi \in \mathcal{K}$; if $b = 1$, then $\mathcal{C}$ chooses a random element from $\mathcal{K}$ as the value for ξ. $\mathcal{C}$ gives $\mathcal{A}$ the test value ξ. Finally, $\mathcal{A}$ outputs a bit b', and wins the game if $b' = b$. $\mathcal{A}$'s advantage in this IND-SK (indistinguishability of shared key) security game is defined to be $\mathrm{Adv}_{\mathcal{A}}^{\mathsf{IND-SK}}(k) = |\Pr[b = b'] - \frac{1}{2}|$. We say that an NIKD-DPKG scheme is secure (IND-SK secure) if for any polynomial-time adversary $\mathcal{A}$ the function $\mathrm{Adv}_{\mathcal{A}}^{\mathsf{IND-SK}}(k)$ is negligible.

[8] Note that the situation that a user with some identity ID_j joins the colluders group is fully covered by allowing the adversary $\mathcal{A}$ to issue EnablePersonalKey queries. $\mathcal{A}$ can compute ID_j's personal key $\kappa_{\mathsf{ID}_j} = \langle f_{\mathsf{ID}_j}, b_{\mathsf{ID}_j}\rangle$ anytime by running algorithm $\mathsf{SetupPersonalKey}$ where the Key-Enabling Centers $\{\mathsf{KEC}_1, ..., \mathsf{KEC}_t\} \backslash \mathsf{IKEC}$ can be impersonated by $\mathcal{A}$ itself (using the master keys of these KECs, which $\mathcal{A}$ knows from the KEC Setup). Only for IKEC, $\mathcal{A}$ does not have the master key. For having the computations done that require $\mathsf{keckey}_{\mathsf{IKEC}}$, $\mathcal{A}$ issues an $\mathsf{EnablePersonalKey}$ query.

4 An NIKD-DPKG Scheme Based on Identity-determined Factors-selection

Now we present the complete scheme for identity-based non-interactive key distribution with distributed personal-key generation sketched in Section 2. The scheme uses the method of identity-determined factors selection introduced there. The five algorithms of the scheme are:

SetupSystem algorithm. Given the security parameter 1^k, the algorithm performs the following steps:

1. Chooses a prime number $q \gtrsim \frac{2}{\pi}2^{2k}$ and another prime number $p \geq \sqrt{2\pi q}+1$ such that $q-1 = c_p p$ where c_p is an even positive integer. In the following, the multiplicative subgroup of order p of $\mathbb{Z}_q^*$ is denoted G_p.
2. Chooses a commutative finite cyclic group G_q of order q such that the most efficient algorithm for solving the generalized discrete-logarithm problem (GDLP, see, e.g., [22]) in G_q has an average computational complexity not smaller than 2^k. For instance, G_q could be the group of points on an elliptic curve over a finite field $E(\mathbb{F}_n)$ (where n is another large prime number of the same bitlength as q) that are generated by some point $B \in E(\mathbb{F}_n)$ which is chosen such that the smallest $c \in \mathbb{N}^*$ for which $cB = O$ is q (see, e.g., [15]). Another instance of G_q is the (multiplicative) subgroup of order q of $\mathbb{Z}_m^*$ where m is a sufficiently large prime number (e.g., of bitlength 1024, for $k = 80$) with $m-1 = c_q q$ where c_q is an even positive integer.
3. Chooses a generator of G_q as base β.
4. Defines a *potential-factors number* $z \in \mathbb{N}^*$ and an *actual-factors number* $h \in \mathbb{N}^*$ such that $\binom{z}{h} \geq 2^{2k}$. The choice of the ratio of z and h is a trade-off between, on one side, keeping the computational complexity of the shared-key derivation low (z increasing, h decreasing from $z/2$) and, on the other side, keeping the size of the personal keys small (z decreasing, h increasing towards $z/2$). Suitable choices are, for instance, $z = 165$ and $h = 82$ (or $z = 256$ and $h = 42$) for $k = 80$, $z = 229$ and $h = 114$ for $k = 112$, $z = 261$ and $h = 130$ (or $z = 1024$ and $h = 44$) for $k = 128$, $z = 389$ and $h = 194$ for $k = 192$, $z = 517$ and $h = 258$ for $k = 256$.
5. Defines a cryptographic hash function (preimage-resistant and collision-resistant, suitable to be modeled as a random oracle [2]) $hid : \{0,1\}^* \longrightarrow \{0,1\}^z$ with the special feature that the number of ones in the resulting bit vector (the Hamming weight) is constantly h. Two examples how such a function can be constructed from a standard hash function are given in Appendix B. Note that the choices of z and h above make the average complexity of finding a collision through random guessing $\geq 2^k$.
6. Defines a cryptographic hash function (preimage-resistant, suitable to be modeled as a random oracle) $hel : G_q \longrightarrow \{0,1\}^{l(k)}$. For instance, if G_q is an elliptic-curve group as described in step 2, hel may return the first $l(k)$ bits of the output of a standard cryptographic hash function (such as SHA [25]) with output length $\geq l(k)$ executed on the concatenation of the binary representations of the two coordinates of the input. If G_q is, as in the alternative described in step 2, a subgroup of the multiplicative group of a prime field, hel may return the first $l(k)$ bits of the output of the standard hash function executed on the binary representation of the input.

The public system parameters are: $\mathsf{params} = \langle p, q, G_q, \beta, z, h, \mathrm{hid}, \mathrm{hel}\rangle$. The shared-key space $\mathcal{K}$ is $\{0,1\}^{l(k)}$.

SetupKEC algorithm. Given params, the algorithm performs the following steps:

1. Randomly chooses a generator of G_p as base α_{KEC}.
2. Generates the *exponents vector* $\mathbf{e}_{\mathsf{KEC}}[1...z] \in (\mathbb{Z}_p^*)^z$ of z different elements $\in \mathbb{Z}_p$, each element being an independently chosen random number.

The KEC's master key is: $\mathsf{masterkey}_{\mathsf{KEC}} = \langle \alpha_{\mathsf{KEC}}, \mathbf{e}_{\mathsf{KEC}}\rangle$.

SetupPersonalKey algorithm. Given params and a bit string $\mathsf{ID} \in \{0,1\}^*$, the algorithm performs the following:

1. Generates ID's *potential-factors vector* $\mathbf{f}_{\mathsf{ID}}[1...z] \in G_p^z$ of z elements, and initializes the elements:
   $\forall i \in \{1, ..., z\} : \ \mathbf{f}_{\mathsf{ID}}[i] \ \leftarrow \ 1$
2. Initializes ID's base: $b_{\mathsf{ID}} \in G_q \ \leftarrow \ \beta$ where β is from the public parameters params.
3. Prepares, using the KECs, one after the other, the two components of ID's personal key:
   $\forall\ \mathrm{KEC}_x \in \{\mathrm{KEC}_1, ..., \mathrm{KEC}_t\}$ [
   (a) Requests from KEC_x (which computes the response by running the algorithm $\mathsf{EnablePersonalKey}$ below) KEC_x's contribution $\mathbf{v}_{[\mathsf{ID},\mathsf{KEC}_x]}$ to $\mathbf{f}_{\mathsf{ID}}$ and the processing of b_{ID} correspondingly:

$$\langle \mathbf{v}_{[\mathsf{ID},\mathsf{KEC}_x]} \in G_p^z,\ b_{\mathsf{ID}}\rangle \ \leftarrow \ \mathrm{KEC}_x.\mathsf{EnablePersonalKey}(\mathsf{ID}, b_{\mathsf{ID}})$$

(b) Checks that b_{ID} is a generator of G_q.[9] Checks for all elements of $\mathbf{v}_{[\mathsf{ID},\mathsf{KEC}_x]}$ that the values are in $\mathbb{Z}_q^*$. If any of these checks failed, then returns without a result but with an alarm that KEC_x has attempted to induce the generation of a leakage-provoking personal key.

(c) Integrates KEC_x's contribution into the potential-factors vector:

$$\forall i \in \{1, ..., z\} : \; \mathbf{f}_{\mathsf{ID}}[i] \; \leftarrow \; \mathbf{f}_{\mathsf{ID}}[i] \; \cdot \; \mathbf{v}_{[\mathsf{ID},\mathsf{KEC}_x]}[i] \mod q$$

where q is from the public parameters params.

Returns ID's personal key $\kappa_{\mathsf{ID}} = \langle b_{\mathsf{ID}}, \mathbf{f}_{\mathsf{ID}} \rangle$.

EnablePersonalKey algorithm. Given params, the $\mathsf{masterkey}_{\mathsf{KEC}}$ of the algorithm executor, the identity whose key shall be enabled $\mathsf{ID} \in \{0,1\}^*$, and its key enabler $b_{\mathsf{ID}} \in G_q$, the algorithm performs the following steps:

1. Computes the ID-specific *selection vector* $\mathbf{s}_{\mathsf{ID}}$, a bit vector of z elements

$$\mathbf{s}_{\mathsf{ID}}[1...z] \in \{0,1\}^z \leftarrow \mathrm{hid}(\mathsf{ID})$$

where *hid* is the special cryptographic hash function defined as part of the public parameters.

2. Computes the ID-specific *selected-exponents sum* $w_{[\mathsf{ID},\mathsf{KEC}]}$

$$w_{[\mathsf{ID},\mathsf{KEC}]} \in \mathbb{Z}_p \; \leftarrow \; \sum_{i=1}^{z} (\mathbf{s}_{\mathsf{ID}}[i] \cdot \mathbf{e}_{\mathsf{KEC}}[i]) \mod p$$

where $\mathbf{e}_{\mathsf{KEC}}$ is from the executor's master key $\mathsf{masterkey}_{\mathsf{KEC}}$, and p is from the public parameters. The multiplication with $\mathbf{s}_{\mathsf{ID}}$[i] is practically a selection of summands from the elements of $\mathbf{e}_{\mathsf{KEC}}$. $w_{[\mathsf{ID},\mathsf{KEC}]}$ is kept secret, especially from ID; it can be deleted after step 6.

3. Randomly picks $r_{[\mathsf{ID},\mathsf{KEC}]}$ from $\mathbb{Z}_p$. This ID-specific "blinding addend" is kept secret, especially from ID; it can be deleted after step 6.

4. Computes the ID-specific "deblinding exponent" $d_{[\mathsf{ID},\mathsf{KEC}]}$

$$d_{[\mathsf{ID},\mathsf{KEC}]} \in G_p \; \leftarrow \; \alpha_{\mathsf{KEC}}^{\left(-h \; \cdot \; r_{[\mathsf{ID},\mathsf{KEC}]} \mod p\right)} \mod q$$

where h is from the public parameters, and α_{KEC} is from the executor's masker key $\mathsf{masterkey}_{\mathsf{KEC}}$. $d_{[\mathsf{ID},\mathsf{KEC}]}$ is kept secret, especially from ID; it can be deleted after step 5.

5. Processes ID's base corresponding to the choice of $r_{[\mathsf{ID},\mathsf{KEC}]}$: $\; b_{\mathsf{ID}} \; \leftarrow \; b_{\mathsf{ID}}^{d_{[\mathsf{ID},\mathsf{KEC}]}}$

6. Generates the KEC's contribution to ID's potential-factors vector $\mathbf{v}_{[\mathsf{ID},\mathsf{KEC}]}[1...z] \in G_p^z$ of z elements, the elements being computed:

$$\forall i \in \{1, ..., z\} : \; \mathbf{v}_{[\mathsf{ID},\mathsf{KEC}]}[i] \; \leftarrow \; \alpha_{\mathsf{KEC}}^{\left(w_{[\mathsf{ID},\mathsf{KEC}]} \; \cdot \; \mathbf{e}_{\mathsf{KEC}}[i] \; + \; r_{[\mathsf{ID},\mathsf{KEC}]} \mod p\right)} \mod q$$

Returns $\langle b_{\mathsf{ID}}, \mathbf{v}_{[\mathsf{ID},\mathsf{KEC}]} \rangle$, i.e., ID's processed base and KEC's contribution to ID's potential-factors vector.

SharedKey algorithm. Given params, the algorithm executor's (i.e., $\mathsf{ID_A}$'s) key $\kappa_{\mathsf{ID_A}} = \langle \mathbf{f}_{\mathsf{ID_A}}, b_{\mathsf{ID_A}} \rangle$ and the other party's identity $\mathsf{ID_B} \in \{0,1\}^*$, the algorithm performs the following steps:

1. Computes the selection vector $\quad \mathbf{s}_{\mathsf{ID_B}}[1...z] \in \{0,1\}^z \leftarrow \mathrm{hid}(\mathsf{ID_B})$

2. Computes the, however "$\mathsf{ID_A}$-specifically blinded", common exponent

$$\lambda_{\mathsf{A,B}} \in G_p \; \leftarrow \; \prod_{i=1}^{z} \mathbf{f}_{\mathsf{ID_A}}[i]^{\; \mathbf{s}_{\mathsf{ID_B}}[i]} \mod q$$

(The exponentiation with $\mathbf{s}_{\mathsf{ID_B}}[i]$ is practically a selection of actual factors from the elements of the potential-factors vector $\mathbf{f}_{\mathsf{ID_A}}$).

[9] Since this check has a significant computational complexity (a regular exponentiation), it is advisable to keep all received (temporary) values of b_{ID} (together with the identity of the producing KEC_x) in a buffer and check just the final value of b_{ID}. Only if the final value fails the test, one must check all buffered values (in the order of their receipt) to find out which KEC_x has (first) delivered an illegal value.

3. Computes the “raw” shared key $\quad \theta_{\mathsf{A,B}} \in G_q \;\leftarrow\; b_{\mathsf{ID_A}}^{\lambda_{\mathsf{A,B}}}$

4. Computes the shared key $\quad \gamma_{\mathsf{A,B}} \in \{0,1\}^{l(k)} \;\leftarrow\; \mathrm{hel}(\theta_{\mathsf{A,B}})$ $\quad$ where *hel* is the cryptographic hash function defined as part of the public parameters.

Returns $\gamma_{\mathsf{A,B}}$

4.1 Consistency

The algorithms of the scheme are consistent if and only if for all pairs of identities $\mathsf{ID_A}, \mathsf{ID_B} \in \{0,1\}^*$ holds: If $\kappa_{\mathsf{ID_A}} = \mathsf{SetupPersonalKey}(\mathsf{params}, \mathsf{ID_A})$, $\kappa_{\mathsf{ID_B}} = \mathsf{SetupPersonalKey}\,(\mathsf{params}, \mathsf{ID_B})$, $\gamma_{\mathsf{A,B}} = \mathsf{SharedKey}(\mathsf{params}, \kappa_{\mathsf{ID_A}}, \mathsf{ID_B})$, and $\gamma_{\mathsf{B,A}} = \mathsf{SharedKey}(\mathsf{params}, \kappa_{\mathsf{ID_B}}, \mathsf{ID_A})$, then $\gamma_{\mathsf{A,B}} = \gamma_{\mathsf{B,A}}$.

$\gamma_{\mathsf{A,B}}$ is the result of function *hel* returned on input $\theta_{\mathsf{A,B}}$ (step 4 of algorithm SharedKey). $\gamma_{\mathsf{B,A}}$ is the result of *hel* returned on input $\theta_{\mathsf{B,A}}$ (step 4 of algorithm SharedKey). The cryptographic hash function *hel* is deterministic, thus if $\theta_{\mathsf{A,B}} = \theta_{\mathsf{B,A}}$ then $\gamma_{\mathsf{A,B}} = \gamma_{\mathsf{B,A}}$. To see that $\theta_{\mathsf{A,B}} = \theta_{\mathsf{B,A}}$, let us first look what $\theta_{\mathsf{A,B}}$ is.
As computed in step 3 of algorithm SharedKey:

$$\theta_{\mathsf{A,B}} \;=\; b_{\mathsf{ID_A}}^{\lambda_{\mathsf{A,B}}}$$

As $b_{\mathsf{ID_A}}$ is computed in algorithm SetupPersonalKey (steps 2 and 3a), which has $b_{\mathsf{ID_A}}$ be processed by all the KECs, each one of them running algorithm EnablePersonalKey:

$$= \left(\beta^{\left(\prod_{x=1}^{t} d_{[\mathsf{ID_A},\mathsf{KEC}_x]} \mod q\right)}\right)^{\lambda_{\mathsf{A,B}}}$$

As $d_{[\mathsf{ID_A},\mathsf{KEC}_x]}$ is computed in step 4 of algorithm EnablePersonalKey:

$$= \left(\beta^{\left(\prod_{x=1}^{t} \alpha_{\mathsf{KEC}_x}^{\left(-h \,\cdot\, r_{[\mathsf{ID_A},\mathsf{KEC}_x]} \mod p\right)} \mod q\right)}\right)^{\lambda_{\mathsf{A,B}}}$$

As $\lambda_{\mathsf{A,B}}$ is computed in step 2 of algorithm SharedKey:

$$= \left(\beta^{\left(\prod_{x=1}^{t} \alpha_{\mathsf{KEC}_x}^{\left(-h \,\cdot\, r_{[\mathsf{ID_A},\mathsf{KEC}_x]} \mod p\right)} \mod q\right)}\right)^{\left(\prod_{i=1}^{z} \mathbf{f}_{\mathsf{ID_A}}[i]^{\,\mathbf{s}_{\mathsf{ID_B}}[i]} \mod q\right)}$$

$$= \beta^{\left(\prod_{x=1}^{t} \alpha_{\mathsf{KEC}_x}^{\left(-h \,\cdot\, r_{[\mathsf{ID_A},\mathsf{KEC}_x]} \mod p\right)} \,\cdot\, \prod_{i=1}^{z} \mathbf{f}_{\mathsf{ID_A}}[i]^{\,\mathbf{s}_{\mathsf{ID_B}}[i]} \mod q\right)}$$

$$= \beta^{\left(X \,\cdot\, \prod_{i=1}^{z} \mathbf{f}_{\mathsf{ID_A}}[i]^{\,\mathbf{s}_{\mathsf{ID_B}}[i]} \mod q\right)} \qquad \text{with } \; X \;=\; \prod_{x=1}^{t} \alpha_{\mathsf{KEC}_x}^{\left(-h \,\cdot\, r_{[\mathsf{ID_A},\mathsf{KEC}_x]} \mod p\right)} \mod q$$

As $\mathbf{f}_{\mathsf{ID_A}}$ is computed in steps 1 and 3c algorithm SetupPersonalKey from the $\mathbf{v}_{[\mathsf{ID_A},\mathsf{KEC}_x]}$'s received from the KECs:

$$= \beta^{\left(X \,\cdot\, \prod_{i=1}^{z}\left(\prod_{x=1}^{t} \mathbf{v}_{[\mathsf{ID_A},\mathsf{KEC}_x]}[i]\right)^{\mathbf{s}_{\mathsf{ID_B}}[i]} \mod q\right)}$$

As $\mathbf{v}_{[\mathsf{ID_A},\mathsf{KEC}_x]}[i]$ is computed in algorithm EnablePersonalKey (step 6):

$$= \beta^{\left(X \,\cdot\, \prod_{i=1}^{z}\left(\prod_{x=1}^{t} \alpha_{\mathsf{KEC}_x}^{\left(w_{[\mathsf{ID_A},\mathsf{KEC}_x]} \,\cdot\, \mathbf{e}_{\mathsf{KEC}_x]}[i] \,+\, r_{[\mathsf{ID_A},\mathsf{KEC}_x]} \mod p\right)} \mod q\right)^{\mathbf{s}_{\mathsf{ID_B}}[i]} \mod q\right)}$$

$$= \beta^{\left(X \,\cdot\, \prod_{i=1}^{z}\left(\prod_{x=1}^{t} \alpha_{\mathsf{KEC}_x}^{w_{[\mathsf{ID_A},\mathsf{KEC}_x]} \,\cdot\, \mathbf{e}_{\mathsf{KEC}_x]}[i] \mod p} \,\cdot\, \prod_{x=1}^{t} \alpha_{\mathsf{KEC}_x}^{r_{[\mathsf{ID_A},\mathsf{KEC}_x]} \mod p}\right)^{\mathbf{s}_{\mathsf{ID_B}}[i]} \mod q\right)}$$

$$= \beta^{\left(X \,\cdot\, Y \,\cdot\, \prod_{i=1}^{z}\left(\prod_{x=1}^{t} \alpha_{\mathsf{KEC}_x}^{r_{[\mathsf{ID_A},\mathsf{KEC}_x]}}\right)^{\mathbf{s}_{\mathsf{ID_B}}[i]} \mod q\right)}$$

$$\text{with } \; Y \;=\; \prod_{i=1}^{z}\prod_{x=1}^{t}\left(\alpha_{\mathsf{KEC}_x}^{w_{[\mathsf{ID_A},\mathsf{KEC}_x]} \,\cdot\, \mathbf{e}_{\mathsf{KEC}_x]}[i] \mod p}\right)^{\mathbf{s}_{\mathsf{ID_B}}[i]} \mod q$$

$$= \beta^{\left(X \,\cdot\, Y \,\cdot\, \prod_{x=1}^{t} \alpha_{\mathsf{KEC}_x}^{\left(r_{[\mathsf{ID_A},\mathsf{KEC}_x]} \,\cdot\, \sum_{i=1}^{z} \mathbf{s}_{\mathsf{ID_B}}[i] \mod p\right)} \mod q\right)}$$

As $\mathbf{s}_{\mathsf{ID_B}}[i]$ is computed in step 1 of algorithm EnablePersonalKey, and according to the definition of the cryptographic hash function *hid* in the algorithm SetupSystem (steps 4 and 5):

$$= \beta^{\left(X \,\cdot\, Y \,\cdot\, \prod_{x=1}^{t} \alpha_{\mathsf{KEC}_x}^{\left(h \,\cdot\, r_{[\mathsf{ID_A},\mathsf{KEC}_x]} \mod p\right)} \mod q\right)}$$

Substituting for X:

$$= \beta^{\left(\prod_{x=1}^{t} \alpha_{\mathsf{KEC}_x}^{\left(-h \,\cdot\, r_{[\mathsf{ID_A},\mathsf{KEC}_x]} \mod p\right)} \,\cdot\, Y \,\cdot\, \prod_{x=1}^{t} \alpha_{\mathsf{KEC}_x}^{\left(h \,\cdot\, r_{[\mathsf{ID_A},\mathsf{KEC}_x]} \mod p\right)} \mod q\right)}$$

$$= \beta^{Y}$$

Substituting for Y:

$$= \beta^{\left(\prod_{i=1}^{z}\prod_{x=1}^{t}\left(\alpha_{\mathsf{KEC}_x}^{\left(w_{[\mathsf{ID_A},\mathsf{KEC}_x]}\cdot \mathbf{e}_{\mathsf{KEC}_x]}[i] \bmod p\right)}\right)^{\mathbf{s}_{\mathsf{ID_B}}[i]} \bmod q\right)}$$

$$= \beta^{\left(\prod_{x=1}^{t}\alpha_{\mathsf{KEC}_x}^{\left(w_{[\mathsf{ID_A},\mathsf{KEC}_x]}\cdot \Sigma_{i=1}^{z}\left(\mathbf{s}_{\mathsf{ID_B}}[i]\cdot \mathbf{e}_{\mathsf{KEC}_x]}[i]\right) \bmod p\right)} \bmod q\right)}$$

As $w_{[\mathsf{ID_B},\mathsf{KEC}_x]}$ is computed in step 2 of algorithm $\mathsf{EnablePersonalKey}$:

$$= \beta^{\left(\prod_{x=1}^{t}\alpha_{\mathsf{KEC}_x}^{\left(w_{[\mathsf{ID_A},\mathsf{KEC}_x]}\cdot w_{[\mathsf{ID_B},\mathsf{KEC}_x]} \bmod p\right)} \bmod q\right)}$$

In the same way we can show that $\theta_{\mathsf{B,A}} = \beta^{\left(\prod_{x=1}^{t}\alpha_{\mathsf{KEC}_x}^{\left(w_{[\mathsf{ID_B},\mathsf{KEC}_x]}\cdot w_{[\mathsf{ID_A},\mathsf{KEC}_x]} \bmod q\right)} \bmod p\right)}$. Obviously, $\theta_{\mathsf{A,B}}$ equals $\theta_{\mathsf{B,A}}$, which means the scheme is consistent.

4.2 Security

An adversary $\mathcal{A}$ in the IND-SK security game in the definition of NIKD-DPKG security in Section 3 can have a non-zero advantage only if it can for at least one tuple $\langle \mathsf{ID_U} \in \{0,1\}^*, \mathsf{ID_V} \in \{0,1\}^*, \gamma_w \in \mathcal{K}\rangle$ where neither $\mathsf{ID_U}$ nor $\mathsf{ID_V}$ have been the argument of an $\mathsf{EnablePersonalKey}$ query and the pair $\mathsf{ID_U}, \mathsf{ID_V}$ has not been the argument of a $\mathsf{RevealSharedKey}$ query tell with a success probability other than 50% whether γ_w is the key shared by $\mathsf{ID_U}$ and $\mathsf{ID_V}$. If *hel* is a random oracle, then the difference between the success probability and $\frac{1}{2}$ is exactly the probability that $\mathcal{A}$ has found out the $\theta_{\mathsf{U,V}}$ corresponding to the pair $\mathsf{ID_U}, \mathsf{ID_V}$. Further, if *hel* is a random oracle then the power to (adaptively) issue $\mathsf{RevealSharedKey}$ queries does not give $\mathcal{A}$ any information that would really help to win the game. The reason is that, if *hel* is a *random* oracle, then $\mathsf{ID_X}$ and $\mathsf{ID_Y}$'s shared-key $\gamma_{\mathsf{X,Y}}$ does not reveal anything about $\theta_{\mathsf{X,Y}}$ or other intermediate results from which $\gamma_{\mathsf{X,Y}}$ was computed.

$$\theta_{\mathsf{A,B}} \text{ is } \beta^{\left(\prod_{\forall \mathsf{KEC}_x\in\{\mathsf{KEC}_1,\ldots,\mathsf{KEC}_t\}} \alpha_{\mathsf{KEC}_x}^{\left(\Sigma_{i=1}^{z}\left(\mathbf{s}_{\mathsf{ID_A}}[i]\cdot \mathbf{e}_{\mathsf{KEC}_x]}[i]\right)\cdot \Sigma_{i=1}^{z}\left(\mathbf{s}_{\mathsf{ID_B}}[i]\cdot \mathbf{e}_{\mathsf{KEC}_x]}[i]\right) \bmod p\right)} \bmod q\right)}$$

$$= \varphi^{\left(\alpha_{\mathsf{IKEC}}^{\left(\Sigma_{i=1}^{z}\left(\mathbf{s}_{\mathsf{ID_A}}[i]\cdot \mathbf{e}_{\mathsf{IKEC}]}[i]\right)\cdot \Sigma_{i=1}^{z}\left(\mathbf{s}_{\mathsf{ID_B}}[i]\cdot \mathbf{e}_{\mathsf{IKEC}]}[i]\right) \bmod p\right)} \bmod q\right)}$$

$$\text{with } \varphi = \beta^{\left(\prod_{\forall \mathsf{KEC}_x\in\{\mathsf{KEC}_1,\ldots,\mathsf{KEC}_t\}\setminus \mathsf{IKEC}} \alpha_{\mathsf{KEC}_x}^{\left(\Sigma_{i=1}^{z}\left(\mathbf{s}_{\mathsf{ID_A}}[i]\cdot \mathbf{e}_{\mathsf{KEC}_x]}[i]\right)\cdot \Sigma_{i=1}^{z}\left(\mathbf{s}_{\mathsf{ID_B}}[i]\cdot \mathbf{e}_{\mathsf{KEC}_x]}[i]\right) \bmod p\right)} \bmod q\right)}$$

$\mathcal{A}$ can easily compute φ because it knows all the master keys needed. In addition, $\mathcal{A}$ can (by issuing $\mathsf{EnablePersonalKey}$ queries) for all ID_j's except $\mathsf{ID_A}, \mathsf{ID_B}$ learn the corresponding pairs $\langle b_{\mathsf{ID}_j}, \mathbf{v}_{\mathsf{ID}_j,\mathsf{IKEC}}\rangle$ where $b_{\mathsf{ID}_j} = \varphi^{\left(\alpha_{\mathsf{IKEC}}^{-h\,\cdot\, r_{\mathsf{ID}_j,\mathsf{IKEC}}}\right)}$ and $\forall i \in \{1,...,z\}$: $\mathbf{v}_{\mathsf{ID}_j,\mathsf{IKEC}}[i] = \alpha_{\mathsf{IKEC}}^{\mathbf{e}_{\mathsf{IKEC}}[i]\,\cdot\,\Sigma_{k=1}^{z}\left(\mathbf{s}_{\mathsf{ID}_j}[k]\cdot \mathbf{e}_{\mathsf{IKEC}}[k]\right) + r_{\mathsf{ID}_j,\mathsf{IKEC}}}$. Obviously, the only unknown variables in the problem to find $\theta_{\mathsf{A,B}}$ are the components of the master key of IKEC and the "blinding addends" $r_{\mathsf{ID}_j,\mathsf{IKEC}}$ randomly chosen by IKEC. Thus, only IKEC appears in the actual problem.

The fact that $\mathcal{A}$ can freely choose the base ϱ for which the response to an $\mathsf{EnablePersonalKey}$ query contains $b_{\mathsf{ID}_j} = \varrho^{\left(\alpha_{\mathsf{IKEC}}^{\left(-h\,\cdot\, r_{[\mathsf{ID},\mathsf{IKEC}]} \bmod p\right)} \bmod q\right)}$ does not make the task to find $\left(\alpha_{\mathsf{IKEC}}^{\left(-h\,\cdot\, r_{[\mathsf{ID},\mathsf{IKEC}]} \bmod p\right)} \bmod q\right)$ easier than if the response always just contained $\beta^{\left(\alpha_{\mathsf{IKEC}}^{\left(-h\,\cdot\, r_{[\mathsf{ID},\mathsf{IKEC}]} \bmod p\right)} \bmod q\right)}$ for a fixed generator β of G_q. $\mathcal{A}$ cannot have multiple (different) bases be exponentiated with $r_{[\mathsf{ID},\mathsf{IKEC}]}$ since $r_{[\mathsf{ID},\mathsf{IKEC}]}$ is for each execution of algorithm $\mathsf{EnablePersonalKey}$ a freshly chosen random number. In both cases, the problem to be solved is the same, namely GDHP in G_q. Likewise, the fact that $\mathcal{A}$ can when responding to a request by $\mathcal{C}$ for the processing of an ID's key-enabler (the request to run algorithm $\mathsf{EnablePersonalKey}$ that is triggered by a $\mathsf{SetupPersonalKey}$ query by $\mathcal{A}$) foist *any* generator of G_q that $\mathcal{A}$ wants on $\mathcal{C}$ as ID's key-enabler b_{ID} does not make the task easier for $\mathcal{A}$: the problem in G_q to be solved is still GDHP. Also the fact that $\mathcal{A}$ knows $\mathsf{ID_A}$ and $\mathsf{ID_B}$'s key-enablers $b_{\mathsf{ID_A}}$ and $b_{\mathsf{ID_A}}$ does not ease the task: the problem in G_q to be solved is still GDHP.

If *hid* is a random oracle, then the power to (adaptively) issue $\mathsf{EnablePersonalKey}$ queries does not make $\mathcal{A}$'s task easier than if $\mathcal{A}$ had been given the n tuples $\langle \mathsf{ID}_1 \in \{0,1\}^*, b_{\mathsf{ID}_1} \in G_q, \mathbf{v}_{\mathsf{ID}_1,\mathsf{IKEC}} \in G_p^z\rangle, ..., \langle \mathsf{ID}_n \in \{0,1\}^*, b_{\mathsf{ID}_n} \in G_q, \mathbf{v}_{\mathsf{ID}_n,\mathsf{IKEC}} \in G_p^z\rangle$ that result from the n $\mathsf{EnablePersonalKey}$ queries all at once, right at the start. The reason is that *hid* is a *random* oracle. Even if $\mathcal{A}$ could identify an $\mathbf{s}_{\mathsf{ID}_j}$ for which the response to an $\mathsf{EnablePersonalKey}$ query with the responding ID_j is more helpful than the response to an $\mathsf{EnablePersonalKey}$ query with another identity ID_k, $\mathcal{A}$ would not have a clue for which identity ID_j it should issue the $\mathsf{EnablePersonalKey}$ query. For the same reason, the power to *choose* the "targets" $\mathsf{ID_A}$ and $\mathsf{ID_B}$ does not make the task easier for $\mathcal{A}$ than if it were given the corresponding pair of selection vectors $\mathbf{s}_{\mathsf{ID_A}}, \mathbf{s}_{\mathsf{ID_B}}$, right at the start. If $\mathcal{A}$ had found another pair $\mathbf{s}_{\mathsf{ID}'_\mathsf{A}}, \mathbf{s}_{\mathsf{ID}'_\mathsf{B}}$ for which it has better chances to guess correctly at the end of the game than for the pair given, this would not help $\mathcal{A}$ since it cannot know the corresponding pair of identities $\mathsf{ID_A}'$ and $\mathsf{ID_B}'$.

Taking all these observations into account, we see that, if *hid* and *hel* are random oracles [2], then winning the game is not easier than (i) finding a collision of *hid*, or (ii) solving the following problem P, i.e., $\mathcal{A}$'s advantage in winning the game in the NIKD security definition with a computational effort of a certain complexity is upper-bounded by the sum of (i) the probability to have found a collision of *hid* after having spent an effort of the same complexity, and (ii) the probability that the best guess of the solution of P which the most efficient algorithm to solve P can make (after a computational effort of the same complexity) is really the solution of P. Problem P is:

Let $q, p, G_q, \beta, \alpha, z, \mathbf{e}$, and h be as defined in the Setup algorithm.

Given are p, q, G_q, β, z, h, and a set of n three-elements-couples, each couple, $1 \leq j \leq n$, consisting of: (i) $\mathbf{s}_j \in \{0,1\}^z$ where $\sum_{k=1}^{z} \mathbf{s}_j[k] = h$, (ii) $b_j \in G_q = \beta^{\left(\alpha^{-h \cdot r_j} \mod q\right)}$, and (iii) $\mathbf{f}_j \in G_p^z$ where $\forall j \in \{1, ..., n\}, \forall i \in \{1, ..., z\}:\ \mathbf{f}_j[i] = \alpha^{\left(r_j\ +\ \mathbf{e}[i] \sum_{k=1}^{z}(\mathbf{e}[k]\ \cdot\ \mathbf{s}_j[k])\right)} \mod q$. Given is furthermore a pair of bit vectors $\mathbf{s}_\mathsf{A} \in \{0,1\}^z$ where $\mathbf{s}_\mathsf{A} \notin \{\mathbf{s}_1, ..., \mathbf{s}_n\}\ \wedge\ \sum_{i=1}^{z} \mathbf{s}_\mathsf{A}[i] = h$, and $\mathbf{s}_\mathsf{B} \in \{0,\ 1\}^z$ where $\mathbf{s}_\mathsf{B} \notin \{\mathbf{s}_1, ..., \mathbf{s}_n\}\ \wedge\ \sum_{i=1}^{z} \mathbf{s}_\mathsf{B}[i] = h$.

Find $\beta^{\left(\alpha^{(\sum_{i=1}^{z}(\mathbf{e}[i]\ \cdot\ \mathbf{s}_\mathsf{A}[i])\ \cdot\ \sum_{i=1}^{z}(\mathbf{e}[i]\ \cdot\ \mathbf{s}_\mathsf{B}[i]))} \mod q\right)}$.

We conjecture that the complexity to solve P is not smaller than the smaller one of the following two values: (i) the complexity to solve GDLP in G_q, and (ii) $\frac{p-1}{2}$. The complexity to solve GDLP in G_q is not greater than $\sqrt{\frac{\pi q}{2}}$ (see Pollard's *rho* algorithm for logarithms [28], and other runtime-efficient algorithms for GDLP, see, e.g., [22]). In step 1 of the Setup algorithm, p was chosen to be $\geq \sqrt{2\pi q} + 1$. Thus, $\frac{p-1}{2}$ (which is $\geq \sqrt{\frac{\pi q}{2}}$) cannot be smaller than the average complexity to solve GDLP in G_q (which is $\leq \sqrt{\frac{\pi q}{2}}$). As the consequence, we simplify our conjecture to: If p and q are chosen as in the Setup of our scheme, then the complexity to solve P is not smaller than the complexity to solve GDLP in G_q. More closely related to the definition of the security of NIKD schemes in Section 3, we conjecture that for any computational complexity C holds: The probability that the best guess which the most efficient algorithm to solve P can make (after an effort of complexity C) is really the solution of a given instance of P is not higher than the probability that the best guess which the most efficient algorithm for GDLP in G_q can make (after an effort of complexity C) is really the solution of a given instance of GDLP in G_q. The reasoning behind this conjecture is similar to an assumption concerning the security of the Digital Signature Algorithm (DSA) and of ECDSA [24]. There (see, e.g., [22] [15]), the signature on a message m is $\langle r, s\rangle = \langle \beta^k,\ k^{-1}(H(m)\ +\ x \cdot r) \mod q\rangle$ where H is the hash function, q is the order of the group, β is a generator of this group, and $x \in \mathbb{Z}_q^*$ is the private key of the signer. Note that the security relies on the impossibility to learn the randomly chosen k. The "blinding factor" k conceals the term $(H(m) + x \cdot r) \mod q$. If somebody who already knows $H(m)$, r, and s could learn k and thus $(H(m)\ +\ x \cdot r) \mod q$, he would effectively have learned the signer's private key x. k is exposed in two different "blinded forms": (i) $k^{-1} \mod q$ is one factor of a (known) product of two factors none of which is revealed separately, and (ii) k is the (unrevealed) exponent of a known power of the known base β. The undisclosed factor α^{r_j} plays in problem P effectively the same role as k for the security of the signature schemes. It has the effect that one cannot retrieve the $\alpha^{\mathbf{e}[i] \sum_{k=1}^{z}(\mathbf{e}[k]\ \cdot\ \mathbf{s}_j[k])} \mod q$ from any given $\mathbf{f}_j[i]$. Also α^{r_j} is exposed in two different "blinded forms": (i) α^{r_j} is one factor in z (known) products of two factors none of which is revealed separately, and (ii) $(\alpha^{r_j})^{-h}$ is the (unrevealed) exponent of a known power of the known base β. If one could learn the factor α^{r_j} (for instance, through first learning $\alpha^{\mathbf{e}[i] \sum_{k=1}^{z}(\mathbf{e}[k]\ \cdot\ \mathbf{s}_j[k])} \mod q$), this would mean that one would effectively have solved the intrinsic GDLP instance "*Given* β^{a_j}*, find* a_j" in G_q where a_j is the $\alpha^{-h \cdot r_j} \mod q$.

Solving P without having learned at least one α^{r_j} requires (i) to solve a system of linear equations in $\mathbb{Z}_p$ where the number of equations n is less than the number of variables in $\mathbb{Z}_p$ in these equations, and then (ii) to choose the right solution. Since the set of solutions of the equations system is not smaller than $p - 1$ and no method for choosing candidates from the set is known that is more efficient than pure guessing, this approach to solve P does not work. Appendix A makes the relation of P to GDLP transparent and provides a detailed argumentation supporting our conjecture that P is not easier than GDLP.

If *hid* and *hel* are random oracles, and (following our conjecture above) the best algorithm for P does (for any computational complexity C) not guess solutions for P more successfully than the best algorithm for GDLP in G_q guesses solutions of GDLP instances in G_q (after computations of the same complexity C), then the advantage of the adversary $\mathcal{A}$ in the game in the NIKD security definition is upper-bounded by the sum of (i) the probability to have found a collision of *hid* (after an effort of the same complexity C), and (ii) the probability that the best algorithm for GDLP in G_q has (after computations of the same complexity C) successfully guessed the solution of a GDLP instance in G_q. Both probabilities are negligible, and so is the sum of both. Thus, also $\mathcal{A}$'s advantage is negligible, and our NIKD scheme is, if *hid* and *hel* are random oracles and our conjecture about the hardness of P holds, secure as defined in Section 3.

5 Conclusion

Non-interactive key distribution *with distributed personal-key generation* (NIKD-DPKG) avoids single points of vulnerability, such as the key generation centers operating with the master key in single-master-key NIKD schemes. NIKD-DPKG also avoids that Big Brother-like powers must be given to a single authority, which could then undetectably and uncontrollably eavesdrop on all communications or tamper with the data transferred. Instead, all authorities in the system must cooperate if they want to retrieve the (session) key shared by a pair of users. The scheme presented provides for non-interactive key distribution with distributed personal-key generation, without any practically relevant restriction on the distribution. NIKD-DPKG can be seen as a generalization of NIKD. NIKD is then the special case where the number of key-enabling centers is 1.

The NIKD-DPKG scheme presented is the fastest practical NIKD scheme so far. Each shared-key generation requires just one fixed-base exponentiation in an ordinary, standard prime-order elliptic-curve group plus (at most) $\approx k$ (non-expensive) multiplications in a subgroup of the multiplicative group of a prime field with a modulus $2k$ bits long. Considering the usual runtime-optimized algorithms for fixed-base exponentiation[10] and taking into account that the order of the elliptic curve group is $2k$ bits long, the average computational effort is essentially k group operations in G_q (typically, point additions in a an elliptic-curve group with field size $2k$ bits long), plus $\approx k$ non-expensive modular multiplications in G_p (p $2k$ bits long) and some computations of negligible costs, such as hashes.

In the introduction, we have compared this with the complexities of today's pairing-based NIKD schemes. In order to put this into the most generally familiar perspective, we finally compare it with the computational complexities of the key-encapsulation/key-decapsulation parts of today's widely used public-key encryption schemes. The key-encapsulation in practical discrete-logarithm based public-key encryption schemes, such as DHIES [1] or ECIES [7], requires two exponentiations (one of them fixed-base) with $2k$-bits exponents, i.e., on average, $4k$ group operations. The key-decapsulation in these schemes requires one (regular) exponentiation with a $2k$-bits exponent, i.e., on average, $3k$ group operations. In comparison, the $\approx k$ modular multiplications in G_p in our scheme do by far not cost as much as the at least $2k$ group operations more in DHIES or ECIES. Evidently, on both sides of an encrypted message transfer the expected running time for the generation of the shared key in our NIKD-DPKG scheme is less than for the encapsulation/decapsulation of the transport key in discrete-logarithm based public-key encryption schemes. Comparing with RSA public-key encryption, the generation of the shared key in our scheme is not as low-cost as the key-encapsulation function in RSA implementations with low-Hamming-weight public-key exponents. However, note that on the other side the key-decapsulation in RSA requires, like in DHIES, a (regular) exponentiation, which, on average, means, just like in DHIES, $3k$ multiplications of operands as big as in DHIES, which makes our scheme on this side significantly faster. We conclude that the method of identity-based factors selection, which underlies the NIKD-DPKG scheme presented, appears to be a useful approach for the construction of very computation-efficient identity-based cryptographic protocols, at least for NIKD-DPKG, including NIKD.

References

1. M. Abdalla, M. Bellare, and P. Rogaway, The oracle Diffie-Hellman assumptions and an analysis of DHIES, *Proc. of Topics in Cryptology - CT-RSA 01*, LNCS vol. 2020, pp. 143–158
2. M. Bellare and P. Rogaway, Random oracles are practical: A paradigm for designing efficient protocols, *Proc. of 1st. ACM Conference on Computer and Communications Security*, ACM 1993, pp. 62–73.
3. K. Bentahar, P. Farshim, J. Malone-Lee, and N. Smart, Generic constructions of identity-based and certificateless KEMs, *Journal of Cryptology*, 2008, vol. 21[2], pp. 178–199.
4. D. Boneh and X. Boyen, Efficient selective identity-based encryption without random oracles, *Journal of Cryptology*, 2011, vol. 24[4], pp. 659–693.
5. D. Boneh and M. Franklin, Identity-based encryption from the Weil pairing, *SIAM Journal on Computing* 3/2003, vol. 32, pp. 586–615.
6. X. Boyen, A tapestry of identity-based encryption: practical frameworks compared, *Int. Journal of Applied Cryptography*, no. 1, 2008, pp. 3–21.
7. Certicom Research: Standards for efficient cryptography, SEC 1: Elliptic Curve Cryptography, Version 2.0, May 21, 2009. http://www.secg.org/download/aid-780/sec1-v2.pdf
8. L. Chen, Z. Cheng, J. Malone-Lee, and N.P. Smart, An efficient ID-KEM based on the Sakai–Kasahara key construction, *IEE Proc. Information Security* vol. 153, March 2006, pp. 19–26.

[10] For instance, precomputing the powers $q_0 = b^{2^0}, q_1 = b^{2^1}, ..., q_{\lfloor \log_2 p \rfloor} = b^{2^{\lfloor \log_2 p \rfloor}}$. For the actual exponentiation at runtime, computing $\prod_{i=0}^{\lfloor \log_2 p \rfloor} q_i^{\mathbf{s}[i]}$, where p is the maximum value of the exponents, b is the base, and $\mathbf{s}[i]$ is the i-th bit of the base-2 representation (starting with 0) of the exponent.

9. L. Chen and C. Kudla, Identity based authenticated key agreement protocols from pairings, *Proc. of 16th IEEE Security Foundations Workshop (2003)*, IEEE Computer Society Press, pp. 219–233.
10. W. Diffie and M. Hellman, New directions in cryptography *IEEE Trans. on Information Theory*, Nov. 1976, vol. 22, pp. 644–654.
11. R. Dupont and A. Enge, Provably secure non-interactive key distribution based on pairings, *Discrete Applied Mathematics* vol. 154, no. 2 (2006), pp. 270–276.
12. D. Fiore and R. Gennaro, Identity-based key-exchange protocols without pairings, *Transactions on Computational Sciences*, special issue on security in computing, part I, 2010, LNCS vol. 6340, pp. 42–77.
13. D. Galindo and F. D. Garcia, A Schnorr-like lightweight identity-based signature scheme. *Proc. of AfricaCrypt 2009*, LNCS vol. 5580, pp. 135–148.
14. R. Gennaro, S. Halevi, H. Krawczyk, T. Rabin, S. Reidt and S.D. Wolthusen, Strongly-resilient and non-interactive hierarchical key-agreement in MANETs, *Proc. of ESORICS 2008*, LNCS vol. 5283, pp. 49–65.
15. D. Hankerson, A. Menezes, and S. Vanstone, *Guide to elliptic curve cryptography*, Springer, 2004.
16. M. Joye and G. Neven (eds.), *Identity-based cryptography*, IOS Press, Cryptology and Information Security Series (CISS), 2009.
17. A. Joux, A one round protocol for tripartite Diffie-Hellman, *Proc. of Fourth Algorithmic Number Theory Symposium*, LNCS vol. 1838, 2000, pp. 385–394.
18. N. Koblitz and A. Menezes, Pairing-based cryptography at high security levels, *Proc. of 10th IMA Int. Conf. on Crypt. & Cod.*, 2005, LNCS vol. 3796, pp. 13–36.
19. B. Lynn, Authenticated identity-based encryption, IACR Cryptology ePrint Archive, Report 2002/072, 2002. http://eprint.iacr.org/2002/072
20. U. Maurer and S. Wolf, The Diffie-Hellman protocol, *Designs, Codes and Cryptography* (Special Issue Public Key Cryptography), vol. 19, no. 3, Nov. 2000, pp. 147–171.
21. U. Maurer and Y. Yacobi, A non-interactive public-key distribution system, *Designs, Codes and Cryptography* vol. 9, no. 3, Nov. 1996, pp. 305–316.
22. A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of applied cryptography*, 5th ed. CRC Press, 2001.
23. Y. Murakami and M. Kasahara, Murakami-Kasahara ID-based key sharing scheme revisited – in comparison with Maurer-Yacobi schemes, IACR Cryptology ePrint Archive, Report 2005/306, 2005. http://eprint.iacr.org/2005/306
24. National Institute for Standards and Technology: *Digital Signature Standard* (DSS). NIST Federal Information Processing Standard FIPS 186-3, 2009.
http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf
25. National Institute for Standards and Technology, *Secure Hash Standard* (SHA). NIST Federal Information Processing Standard FIPS 180-3, 2008.
http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf
26. E. Okamoto and K. Tanaka, Key distribution based on identification information, *IEEE Journal on Selected Areas in Communications*, vol. 7, no. 4, 1989, pp. 481–485.
27. K. G. Paterson and S. Srinivasan, On the relations between non-interactive key distribution, identity-based encryption and trapdoor discrete log groups, *Designs, Codes and Cryptography* vol. 52, no. 2, Aug. 2009, pp. 219–241.
28. J. M. Pollard, Monte Carlo methods for index computation (mod q), *Mathematics of Computation* vol. 32 (1978), pp. 918–924.
29. R. Sakai, K. Ohgishi and M. Kasahara, Cryptosystems based on pairing, Symp. on Cryptography and Information Security, Okinawa, Japan, Jan. 26–28, 2000.
30. O. Schirokauer, Discrete logarithms and local units, *Philosophical Transactions of the Royal Society of London* (A) 345 (1993), pp. 409–423.
31. C.P. Schnorr, Efficient signature generation by smart cards, *Journal of Cryptology*, 1991, vol. 4[3], pp. 161–174.
32. A. Shamir, Identity-based cryptosystems and signature schemes, *Proc. of CRYPTO 84*, LNCS vol. 196, pp. 47–53.
33. N. Smart, An identity based authenticated key agreement protocol based on the Weil pairing, *Electronics Letters* vol. 38 (2002), pp. 630–632.

Appendix

A. The Problem P

In the random oracle model, breaking the security of the scheme for non-interactive key distribution with distributed personal-key generation (NIKD-DPKG) presented in Section 4 is at least as hard as solving problem P or breaking the collision resistance of the cryptographic hash function *hid*. We conjecture that (for p and q chosen as in the setup algorithm of the scheme) P is as hard as the generalized discrete logarithm problem in the group G_q in the NIKD-DPKG scheme. In the following, we show how P relates to the generalized discrete logarithm problem and we provide evidence supporting our conjecture. Step by step, we introduce a series of problems, finally getting to P.

We start with recalling the discrete logarithm problem (see, e.g., [22]) in its two usual embodiments. The original discrete logarithm problem (DLP) in prime-order groups[11] is: Let G_p be the multiplicative subgroup of order p of $\mathbb{Z}_q^*$ where both p and q are prime numbers. Let α be a generator of G_p. Given $p,\ q,\ \alpha$, and $\alpha^a \mod q$, find a.[12] The generalized discrete logarithm problem (GDLP) in prime-order groups is: Let G_q be a finite cyclic group of prime order q and let β be a generator of G_q. Given q, G_q, β, and β^a, find a.

The next problem resembles DLP. Let G_p and α be as above. The *Secret-base discrete logarithm problem* (SBDLP) is: Given $p,\ q$, and $\alpha^a \mod q$, find a. Concerning the hardness of the problem, first note that, different to DLP, the base α is *not* revealed. For any $\alpha^a \mod q$ other than 1 there are $p-1$ different a_τ's $\in \mathbb{Z}_p^*$ for which an $\alpha_\tau \in G_p$ exists such that $\alpha_\tau^{a_\tau} \equiv \alpha^a \pmod q$. Each pair $\langle a_\tau, \alpha_\tau \rangle$ looks equally likely to be the one with a. An algorithm for finding a without knowing α can only deliver a result that has the *probability* $\frac{1}{p-1}$ that it *really* is a. This is crucially different from DLP, where for each solution candidate $a_\tau \in \mathbb{Z}$ it can easily be checked whether the candidate really *is* a solution, and for which even algorithms of subexponential complexity, such as the adaptation of the Number Field Sieve for DLP [30], and other algorithms of significantly lesser complexity than $\frac{p-1}{2}$ (the average complexity of brute-force testing), such as Pollard's *rho* algorithm for logarithms [28], are known. All these algorithms require the knowledge of the base α. None of these algorithms solves SBDLP.

Further below, we will introduce two variations of the Diffie-Hellman problem. First, we recall the problem in its two usual embodiments. The (original, specific) Diffie-Hellman problem (DHP, see [10] or, e.g., [22]) in prime-order groups is: Let G_p be the multiplicative subgroup of order p of $\mathbb{Z}_q^*$ where both p and q are prime numbers. Let α be a generator of G_p. Given $p,\ q,\ \alpha,\ \alpha^a \mod q$ and $\alpha^b \mod q$, find $\alpha^{ab} \mod q$. The Generalized Diffie-Hellman problem (GDHP, see, e.g., [20] or [22]) in prime-order groups is: Let G_q be a finite cyclic group of prime order q and let β be a generator of G_q. Given $q, G_q, \beta,\ \beta^a$ and β^b, find β^{ab}.

A problem essentially identical with GDHP is the following. Let G_q and β be as in GDHP. Let u and v be elements of $\mathbb{Z}_q^*$. The *Blinded-exponents-revealed GDHP* (BERGDHP) is: Given $q,\ G_q,\ \beta$, and two couples $\langle \beta^{\left(\frac{1}{u} \mod q\right)},\ au \mod q\rangle$ and $\langle \beta^{\left(\frac{1}{v} \mod q\right)},\ bv \mod q\rangle$, find β^{ab}. BERGDHP is computationally equivalent to GDHP. An algorithm that solves GDHP can provide the core function of an algorithm that solves BERGDHP: Given $q,\ G_q,\ \beta$, and the two couples $\langle \beta^{\left(\frac{1}{u} \mod q\right)},\ au \mod q\rangle$ and $\langle \beta^{\left(\frac{1}{v} \mod q\right)},\ bv \mod q\rangle$, it first computes $\beta^a \leftarrow \left(\beta^{\left(\frac{1}{u} \mod q\right)}\right)^{(au \mod q)}$ and $\beta^b \leftarrow \left(\beta^{\left(\frac{1}{v} \mod q\right)}\right)^{(bv \mod q)}$. Then it uses the algorithm for GDHP, with the two intermediate results as the input. An algorithm that solves BERGDHP can provide the core function of an algorithm that solves GDHP: Given $q,\ G_q,\ \beta,\ x = \beta^a$ and $y = \beta^b$, it first arbitrarily picks $s = au \mod q$ and $t = bv \mod q$ from $\mathbb{Z}_q^*$, computes then $\rho \leftarrow x^{\frac{1}{s} \mod q}\ (= \beta^{\frac{1}{u} \mod q})$ and $\sigma \leftarrow y^{\frac{1}{t} \mod q}\ (= \beta^{\frac{1}{v} \mod q})$, and finally uses the algorithm for BERGDHP with the two couples $\langle \rho, s\rangle$ and $\langle \sigma, t\rangle$ as the input.

Now we generalize BERGDHP. Let G_q and β be as in BERGDHP. Let p be a prime factor of $(q-1)$, and let G_p be the multiplicative subgroup of order p of $\mathbb{Z}_q^*$. The *Many-arguments exponent-powers BERGDHP* (MBP) is: Given are $q,\ G_q,\ \beta,\ p$, a set of n different couples $\{\ \langle \beta^{\left(\frac{1}{y_1} \mod q\right)},\ a_1 y_1 \mod q\rangle,\ ...,\ \langle \beta^{\left(\frac{1}{y_n} \mod q\right)},\ a_n y_n \mod q\rangle\ \},\ \forall j \in \{1, ..., n\} : a_j, y_j \in G_p$, and n exponent-exponents $\{c_1, ..., c_n\},\ \forall j \in \{1, ..., n\} : c_j \in \{0, ..., p-1\}$. Find $\beta^{\prod_{j=1}^{n} a_j^{c_j}}$. Obviously, MBP is at least as hard as BERGDHP (and thus as GDHP).

In the following variant of MBP, the c_j's are not prescribed. Let G_q, β, and G_p be as above. The *Existential MBP* (EMBP) is: Given are q, G_q, β, p, and a set of n different couples $\{\langle \beta^{\left(\frac{1}{y_1} \mod q\right)},\ a_1 y_1 \mod q\rangle,\ ...,\ \langle \beta^{\left(\frac{1}{y_n} \mod q\right)},\ a_n y_n \mod q\rangle\ \}, \forall j \in \{1, ..., n\} : a_j, y_j \in G_p$. The task is: Choose a tuple of n exponent-exponents $\langle c_1, ..., c_n\rangle,\ \forall j \in \{1, ..., n\} : c_j \in \{0, ..., p-1\}$, at least two of them non-zero, and find the corresponding $\beta^{\prod_{j=1}^{n} a_j^{c_j}}$.

[11] We look here only at groups of prime order.

[12] The order of the group is, although often not explicitly given, in most DLP instances revealed as part of the specification of the group. We just state it explicitly.

Note that even the easiest solutions still mean solving an instance of BERGDHP in G_q. Thus, we conjecture that also EMBP is at least as hard as BERGDHP (and thus as GDHP).

The next problem resembles DHP. But here, the base is not revealed, just as in SBDLP above. Let G_p be the multiplicative subgroup of order p of $\mathbb{Z}_q^*$ where both p and q are prime numbers. Let α be a generator of G_p. The *Secret-base Diffie-Hellman problem* (SBDHP) is: Given $p,\ q,\ \alpha^a \mod q,$ and $\alpha^b \mod q$, find $\alpha^{ab} \mod q$. Concerning the hardness of the problem, note that for any given pair $\langle \alpha^a \mod q,\ \alpha^b \mod q\rangle$ where none of the two elements is 1 there are $p-1$ different tuples $\langle \alpha_\tau, a_\tau, b_\tau\rangle$'s (resulting in $p-1$ different values $\alpha_\tau^{a_\tau b_\tau} \mod q$) for which $\alpha_\tau^{a_\tau} \equiv \alpha^a \pmod q \wedge\ \alpha_\tau^{b_\tau} \equiv \alpha^b \pmod q$. Each element of G_p other than 1 looks equally likely to be α^{ab} mod q. An algorithm to compute $\alpha^{ab} \mod q$ without knowing α can only deliver a result that has the *probability* $\frac{1}{p-1}$ that it really *is* $\alpha^{ab} \mod q$. This is crucially different from DHP which can be solved using an algorithm that solves DLP, adding only polynomial complexity, and which is, because of the existence of efficient algorithms for DLP, just of *subexponential* complexity. The (efficient) algorithms for DLP require the knowledge of the base α. They *do not* help solving SBDHP. Even if an algorithm intended to solve SBDHP could use an oracle which, when queried with three group elements $\langle \alpha^\rho \mod q,\ \alpha^\sigma \mod q,\ c\rangle$, tells whether $c = \alpha^{\rho\sigma} \mod q$ or not, the average computational complexity of this algorithm would still be the average computational complexity of pure guessing (i.e., $\frac{p-1}{2}$), which means, the complexity would be fully exponential in the bit length of p.

The problem introduced now is a generalization of SBDHP. Let G_p and α be as in SBDHP. The *Subset-sums exponents SBDHP* (SSESBDHP) is: Given $p, q,$ and a set of z group elements $\{\alpha^{e_1} \mod q, ..., \alpha^{e_z} \mod q\}$, and two non-empty subsets $A = \{a_1, ..., a_x\}$ and $B = \{b_1, ..., b_y\}$ of $\{1, ..., z\}$, find $\alpha^{\left(\sum_{i=1}^{x} e_{a_i}\ \cdot\ \sum_{i=1}^{y} e_{b_i}\right)} \mod q$. Obviously, SSESBDHP is at least as hard as SBDHP.

SSESBDHP may be be formulated differently. First, consider that the subsets A and B above can as well be expressed as indexed arrays (for short: *vectors*) of YES/NO (1/0) elements which select the e_{a_i}'s and e_{b_i}'s from a z-elements vector $\mathbf{e} \in \mathbb{Z}_p^z$. Then, the problem can be expressed as follows. Let G_p and α be as in SBDHP, and let $\mathbf{e}$ be a vector of z elements of $\mathbb{Z}_p$. Given are $p, q, z,$ and $\mathbf{f} \in G_p^z$, a vector of z elements in G_p where $\forall i \in \{1, ..., z\}:\ \mathbf{f}[i] = \alpha^{\mathbf{e}[i]} \mod q$. Given is furthermore a pair of bit vectors $\mathbf{s}_\mathsf{A} \in \{0,1\}^z$ and $\mathbf{s}_\mathsf{B} \in \{0,1\}^z$. Find $\alpha^{\left(\sum_{i=1}^{z}(\mathbf{e}[i]\ \cdot\ \mathbf{s}_\mathsf{A}[i])\ \cdot\ \sum_{i=1}^{z}(\mathbf{e}[i]\ \cdot\ \mathbf{s}_\mathsf{B}[i])\right)} \mod q$.[13]

In the following problem, $\mathbf{f}$ is revealed only indirectly, but in multiple embodiments. Let $G_p, \alpha, z,$ and $\mathbf{e}$ be as in SSESBDHP. The *Private-factors SSESBDHP* (PFSSESBDHP) is: Given are $p, q, z,$ and a set of n couples, each one, $1 \le j \le n$, consisting of: (i) $\mathbf{s}_j \in \{0,1\}^z$ and (ii) $\mathbf{f}_j \in G_p^z$ where $\forall j \in \{1, ..., n\}, \forall i \in \{1, ..., z\}$: $\mathbf{f}_j[i] = \alpha^{\mathbf{e}[i] \sum_{k=1}^{z}(\mathbf{e}[k]\ \cdot\ \mathbf{s}_j[k])} \mod q$. Given is furthermore a pair of bit vectors $\mathbf{s}_\mathsf{A}$ and $\mathbf{s}_\mathsf{B} \in \{0,\ 1\}^z$, both vectors being linearly independent from $\{\mathbf{s}_1, ..., \mathbf{s}_n\}$. Find $\alpha^{\left(\sum_{i=1}^{z}(\mathbf{e}[i]\ \cdot\ \mathbf{s}_\mathsf{A}[i])\ \cdot\ \sum_{i=1}^{z}(\mathbf{e}[i]\ \cdot\ \mathbf{s}_\mathsf{B}[i])\right)} \mod q$. Concerning the hardness, note that there are at least $p-1$ different tuples $\langle \alpha_\tau, \mathbf{e}_\tau\rangle$ which satisfy $\forall j \in \{1...n\}, \forall i \in \{1...z\}$: $\alpha_\tau^{\mathbf{e}_\tau[i] \sum_{k=1}^{z}(\mathbf{e}_\tau[k]\ \cdot\ \mathbf{s}_j[k])} \mod q = \mathbf{f}_j[i]$. These tuples result in $p-1$ different values $\alpha_\tau^{\left(\sum_{i=1}^{z}(\mathbf{e}_\tau[i]\ \cdot\ \mathbf{s}_\mathsf{A}[i])\ \cdot\ \sum_{i=1}^{z}(\mathbf{e}_\tau[i]\ \cdot\ \mathbf{s}_\mathsf{B}[i])\right)}$ mod q. Each value looks equally likely to be $\alpha^{\left(\sum_{i=1}^{z}(\mathbf{e}[i]\ \cdot\ \mathbf{s}_\mathsf{A}[i])\ \cdot\ \sum_{i=1}^{z}(\mathbf{e}[i]\ \cdot\ \mathbf{s}_\mathsf{B}[i])\right)} \mod q$. An algorithm to compute the latter without knowing α or $\mathbf{e}$ can only deliver a result that has the *probability* $\frac{1}{p-1}$ that it really *is* this value. This means, even with access to an oracle (which when given $\mathbf{s}_\mathsf{A}, \mathbf{s}_\mathsf{B}$, and some $c \in G_p$ tells whether c is the solution) the complexity would be $\frac{p-1}{2}$, i.e., fully exponential in the bit length of p.

The (computationally equivalent) next problem stretches over two (related) groups. Let $p, q, G_p, \alpha, z,$ and $\mathbf{e}$ be as in PFSSESBDHP. Let G_q be a finite cyclic group of prime order q and let β be a generator of G_q. The *Exponentiated PFSSESBDHP* (EPFSSESBDHP) is: Given are $p, q, G_q, \beta, z,$ and a set of n couples, each one, $1 \le j \le n$, consisting of: (i) $\mathbf{s}_j \in \{0,1\}^z$ and (ii) $\mathbf{f}_j \in G_p^z$ where $\forall j \in \{1, ..., n\}, \forall i \in \{1, ..., z\}:\ \mathbf{f}_j[i] = \alpha^{\mathbf{e}[i] \sum_{k=1}^{z}(\mathbf{e}[k]\ \cdot\ \mathbf{s}_j[k])}$ mod q. Given is furthermore a pair of bit vectors $\mathbf{s}_\mathsf{A}$ and $\mathbf{s}_\mathsf{B} \in \{0,\ 1\}^z$, both being linearly independent from $\{\mathbf{s}_1, ..., \mathbf{s}_n\}$. Find $\beta^{\left(\alpha^{(\sum_{i=1}^{z}(\mathbf{e}[i]\ \cdot\ \mathbf{s}_\mathsf{A}[i])\ \cdot\ \sum_{i=1}^{z}(\mathbf{e}[i]\ \cdot\ \mathbf{s}_\mathsf{B}[i]))} \mod q\right)}$. Concerning the hardness of the problem, note that the exponentiation of β is an injective map from G_p to G_q, and note further that the given information about G_q and β does not help to find $\alpha^{\left(\sum_{i=1}^{z}(\mathbf{e}[i]\ \cdot\ \mathbf{s}_\mathsf{A}[i])\ \cdot\ \sum_{i=1}^{z}(\mathbf{e}[i]\ \cdot\ \mathbf{s}_\mathsf{B}[i])\right)} \mod q$. Thus, EPFSSESBDHP is as hard as PFSSESBDHP.

In the next problem, the elements of the $\mathbf{f}_j$'s are revealed only multiplicatively blinded. Let $p, q, G_q, \beta, G_p, \alpha, z,$ and $\mathbf{e}$ be as in EPFSSESBDHP. The *Blinded private-factors EPFSSESBDHP* (BFEPFSSESBDHP) is: Given are $p, q, G_q, z,$ and a set of n three-elements-couples, each couple, $1 \le j \le n$, consisting of: (i) $\mathbf{s}_j \in \{0,1\}^z$, (ii) $\mathbf{b}_j \in G_q^{z+1}$ where $\forall j \in \{1, ..., n\}, \forall h \in \{0, ..., z\}:\ \mathbf{b}_j[h] = \beta^{\left(\alpha^{-h \cdot r_j} \mod q\right)}$, and (iii) $\mathbf{f}_j \in G_p^z$ where $\forall j \in \{1, ..., n\}, \forall i \in \{1, ..., z\}:\ \mathbf{f}_j[i] = \alpha^{\left(r_j\ +\ \mathbf{e}[i] \sum_{k=1}^{z}(\mathbf{e}[k]\ \cdot\ \mathbf{s}_j[k])\right)} \mod q$. Given is furthermore a pair of bit vectors $\mathbf{s}_\mathsf{A}$ and $\mathbf{s}_\mathsf{B} \in \{0,\ 1\}^z$, none of them an element of $\{\mathbf{s}_1, ..., \mathbf{s}_n\}$. Find $\beta^{\left(\alpha^{(\sum_{i=1}^{z}(\mathbf{e}[i]\ \cdot\ \mathbf{s}_\mathsf{A}[i])\ \cdot\ \sum_{i=1}^{z}(\mathbf{e}[i]\ \cdot\ \mathbf{s}_\mathsf{B}[i]))} \mod q\right)}$.

[13] The multiplications with $\mathbf{s}_\mathsf{A}[i]$, $\mathbf{s}_\mathsf{B}[i]$ are effectively selections of summands from $\mathbf{e}$, determined by the values of the elements of the bit vectors $\mathbf{s}_\mathsf{A}$ and $\mathbf{s}_\mathsf{B}$.

Concerning the hardness of the problem, two cases can be distinguished: (i) $\mathbf{s}_{\mathsf{A}}$ and $\mathbf{s}_{\mathsf{B}}$ are both linearly independent in $\mathbb{Z}_p$ from $\{\mathbf{s}_1, ..., \mathbf{s}_n\}$, and (ii) at least one of the two is a linear combination in $\mathbb{Z}_p$ of $\{\mathbf{s}_1, ..., \mathbf{s}_n\}$. An algorithm $\mathcal{A}$ which solves BFEPFSSESBDHP in case (i) can be used to solve EPFSSESBDHP as follows: Let all r_j be 0, i.e., assign to all elements of all $\mathbf{b}_j$'s the value β, and let all $\mathbf{f}_j$'s of the BFEPFSSESBDHP instance be the $\mathbf{f}_j$'s of the FEPFSSESBDHP instance. Then use $\mathcal{A}$ to get the solution of this BFEPFSSESBDHP instance, which is also the solution of the EPFSSESBDHP instance. In case (i), BFEPFSSESBDHP is obviously as hard as EPFSSESBDHP. Even with access to an oracle which when given $\mathbf{s}_{\mathsf{A}}, \mathbf{s}_{\mathsf{B}}$, and some $c \in G_q$ tells whether c is the solution, the complexity would be $\frac{p-1}{2}$, i.e., fully exponential in the bit length of p. Case (ii) can be seen as an instance of EMBP (see above). Let's assume it is $\mathbf{s}_{\mathsf{A}}$ which is a linear combination in $\mathbb{Z}_p$ of $\{\mathbf{s}_1, ..., \mathbf{s}_n\}$. Let h_{B} then be the Hamming weight of $\mathbf{s}_{\mathsf{B}}$. For each j, the y_j of the EMBP instance is $\alpha^{h_{\mathsf{B}} \cdot r_j} \mod q$, and the $a_j y_j \mod q$ of the EMBP instance is $\prod_{i=1}^{z} (\mathbf{f}_j[i])^{\mathbf{s}_{\mathsf{B}}[i]} \mod q$. Then, the solution of BEPFSSESBDHP is a solution of this EMBP instance, where the exponent-exponents $\langle c_1, ..., c_n \rangle$ are the $c_j \in \mathbb{Z}_p$ that solve the following vector congruence: $\sum_{j=1}^{n} (c_j \mathbf{s}_j) \equiv \mathbf{s}_{\mathsf{A}} \pmod p$. Likewise, if $\mathbf{s}_{\mathsf{B}}$ instead of $\mathbf{s}_{\mathsf{A}}$ is the linear combination of $\{\mathbf{s}_1, ..., \mathbf{s}_n\}$, then the solution of BEPFSSESBDHP is a solution of the EMBP instance where for each j, the y_j is $\alpha^{h_{\mathsf{A}} \cdot r_j} \mod q$ (h_{A} being the Hamming weight of $\mathbf{s}_{\mathsf{A}}$), and the $a_j y_j \mod q$ is $\prod_{i=1}^{z} (\mathbf{f}_j[i])^{\mathbf{s}_{\mathsf{A}}[i]} \mod q$ (the $\langle c_1, ..., c_n \rangle$ are here the c_j's that solve the congruence $\sum_{j=1}^{n} (c_j \mathbf{s}_j) \equiv \mathbf{s}_{\mathsf{B}} \pmod p$). However, it should be noted that in BEPFSSESBDHP there is some information available that is not available in EMBP. First, the value $\beta^{\left(\frac{1}{y_j}\right)^h \mod q}$ is given not only for $h = 1$ but for all $h \in \{0, ..., z\}$. Second, the j (unrevealed) a_j's $(= \alpha^{\left(\sum_{i=1}^{z}(\mathbf{e}[i] \cdot \mathbf{s}_j[i]) \cdot \sum_{i=1}^{z}(\mathbf{e}[i] \cdot \mathbf{s}_{\mathsf{A}}[i])\right)} \mod q)$ are not independently chosen random numbers but have certain (partly known) relations between them. Thus, we are not able to reduce EMBP to BEPFSSESBDHP in case (ii). Nevertheless, we note that there is no apparent way how the additional elements of the $\mathbf{b}_j$'s or the knowledge of some *relations* between the $(\alpha^{\left(\sum_{i=1}^{z}(\mathbf{e}[i] \cdot \mathbf{s}_j[i]) \cdot \sum_{i=1}^{z}(\mathbf{e}[i] \cdot \mathbf{s}_{\mathsf{A}}[i])\right)} \mod q)$'s can ease the problem. Another observation that at first sight may appear as a hint to an efficient method to solve BEPFSSESBDHP in case (ii) is that one can divide (in G_q) one element of $\mathbf{f}_j$ (say $\mathbf{f}_j[i]$) by another element of $\mathbf{f}_j$ (say $\mathbf{f}_j[l]$), in this way getting values such as $\alpha^{\left((\mathbf{e}[i] - \mathbf{e}[l]) \cdot \sum_{k=1}^{z}(\mathbf{e}[k] \cdot \mathbf{s}_j[k])\right)} \mod q$ that are not (multiplicatively) blinded with $\alpha^{r_j} \mod q$ anymore. From such values, one can (if $\mathbf{s}_{\mathsf{A}}$ is the linear combination of $\{\mathbf{s}_1, ..., \mathbf{s}_n\}$) compute the corresponding values $\alpha^{\left((\mathbf{e}[i] - \mathbf{e}[l]) \cdot \sum_{k=1}^{z}(\mathbf{e}[k] \cdot \mathbf{s}_{\mathsf{A}}[k])\right)} \mod q$ $(= \alpha^{\left(\mathbf{e}[i] \sum_{k=1}^{z}(\mathbf{e}[k] \cdot \mathbf{s}_{\mathsf{A}}[k])\right)} \cdot$ $\cdot\, \alpha^{-\left(\mathbf{e}[l] \sum_{k=1}^{z}(\mathbf{e}[k] \cdot \mathbf{s}_{\mathsf{A}}[k])\right)} \mod q)$. However, not different from $\alpha^{r_j} \mod q$, also $\alpha^{-\left(\mathbf{e}[l] \sum_{k=1}^{z}(\mathbf{e}[k] \cdot \mathbf{s}_{\mathsf{A}}[k])\right)} \mod q$ is a "blinding factor" one cannot eliminate without replacing it with another one. Considering that no approach is apparent which may ease BEPFSSESBDHP in case (ii), we conjecture that BEPFSSESBDHP in case (ii) is as hard as EMBP, and finally as hard as GDLP in G_q. Both cases (i) and (ii) combined, we conjecture that the complexity of algorithms to solve BEPFSSESBDHP has a lower bound in the lower one of the following two values: (i) the complexity to solve GDLP in G_q, and (ii) $\frac{p-1}{2}$.

In the last problem, all *selection vectors* $\mathbf{s}_j, \mathbf{s}_{\mathsf{A}}$, and $\mathbf{s}_{\mathsf{B}}$ have the same Hamming weight. Let $p, q, G_q, \beta, G_p, \alpha, z$, and $\mathbf{e}$ be as in BFEPFSSESBDHP. Let h be an element of $\{1, ..., z-1\}$. The *Predetermined Hamming weight BFEPFSSESBDHP* (P) is: Given are p, q, G_q, β, z, h, and a set of n three-elements-couples, each couple, $1 \leq j \leq n$, consisting of: (i) $\mathbf{s}_j \in \{0,1\}^z$ where $\sum_{k=1}^{z} \mathbf{s}_j[k] = h$, (ii) $b_j \in G_q = \beta^{(\alpha^{-h \cdot r_j} \mod q)}$, and (iii) $\mathbf{f}_j \in G_p^z$ where $\forall j \in \{1, ..., n\}, \forall i \in \{1, ..., z\} : \mathbf{f}_j[i] = \alpha^{\left(r_j + \mathbf{e}[i] \sum_{k=1}^{z}(\mathbf{e}[k] \cdot \mathbf{s}_j[k])\right)} \mod q$. Given is furthermore a pair of bit vectors $\mathbf{s}_{\mathsf{A}} \in \{0,1\}^z$ where $\mathbf{s}_{\mathsf{A}} \notin \{\mathbf{s}_1, ..., \mathbf{s}_n\} \wedge \sum_{i=1}^{z} \mathbf{s}_{\mathsf{A}}[i] = h$, and $\mathbf{s}_{\mathsf{B}} \in \{0, 1\}^z$ where $\mathbf{s}_{\mathsf{B}} \notin \{\mathbf{s}_1, ..., \mathbf{s}_n\} \wedge \sum_{i=1}^{z} \mathbf{s}_{\mathsf{B}}[i] = h$. Find $\beta^{\left(\alpha^{(\sum_{i=1}^{z}(\mathbf{e}[i] \cdot \mathbf{s}_{\mathsf{A}}[i]) \cdot \sum_{i=1}^{z}(\mathbf{e}[i] \cdot \mathbf{s}_{\mathsf{B}}[i]))} \mod q\right)}$. If the value $\beta^{\left(\frac{1}{\alpha_j^r}\right)^t \mod q}$ were given not only for $t = h$ but for all $t \in \{0, ..., z\}$, then P would just be a special case of BFEPFSSESBDHP. There is no sign that these special instances of BFEPFSSESBDHP are easier than the average instance of BFEPFSSESBDHP.[14] P equals these special cases, only that *less* information is revealed. Thus, we conjecture that also the complexity of P has a lower bound in the lower one of the following two values: (i) the complexity of GDLP in G_q, and (ii) $\frac{p-1}{2}$.

[14] P may be *harder* since for each j only *one* element of $\mathbf{b}_j$, namely $\beta^{\left(\left(\frac{1}{\alpha^{r_j}}\right)^h \mod q\right)}$, is revealed.

B.1 Hash Function with Predetermined Hamming Weight

A cryptographic hash function which produces results of a constant, predetermined Hamming weight can be constructed from a standard cryptographic hash function, such as SHA [25], using the following algorithm. The idea is to initially set all bits of the result string to 0, and then switch single bits to 1, the latter as often as the desired value of the Hamming weight requires. Which bits are selected to be switched to 1 is essentially determined by the output of the standard cryptographic hash function. Two pseudo-random selectors are read from this output. The first one is an offset. The second one determines which appearance of 0 after the offset is switched to 1 (the indices of the bits being treated as an additive finite group).

Given the bit string $input \in \{0,1\}^*$ to be hashed, the bit length $z \in \mathbb{N}^*$ of the bit string to be produced, the required Hamming weight $h \in \{0,...,z\}$, and the standard cryptographic hash function H of output bit length $hlen \in \mathbb{N}^*$, the algorithm performs the following steps:

```
 1: Determine the number of bits needed to represent the indices $(0,...,z-1)$:
    $w \in \mathbb{N}^* \leftarrow \lceil \log_2 z \rceil$
 2: Determine the bit lengths of the *raw* (uncut) result bit string *rawres*:
    $n \in \mathbb{N}^* \leftarrow 2^w$
 3: Determine the maximum number of "pseudo-random" bits needed:
    $m \in \mathbb{N}^* \leftarrow (n - z + h) \cdot w \cdot 2$

    /* Produce a bit string *src* as a source of "pseudo-random" indices */
 4: for $(bnr \in \mathbb{N}^* \leftarrow 0,\ src \in \{0,1\}^* \leftarrow$ "",  $ims \in \{0,1\}^* \leftarrow input;\ bnr < m;\ bnr \leftarrow bnr + hlen)$ do
 5:     $hims \in \{0,1\}^{hlen} \leftarrow H(ims)$
 6:     Append *hlen* bits to the source of "pseudo-random" indices:
        $src \leftarrow src \,||\, hims$
 7:     Create a fresh input string for the next call of $H$:
        $ims \leftarrow ims \,||\, hims$
 8: end for

 9: Create the *raw* (uncut) result bit string *rawres* of bit length $n$ (the index running from 0 to $(n-1)$) and set
    all its bits to 0
10: Initialize the counter of bits switched to 1:
    $ones \in \mathbb{N}^0 \leftarrow 0$
11: Initialize the pointer from where to read the next pair of "pseudo-random" indices from the bit string *src*:
    $p \in \mathbb{N}^0 \leftarrow 0$
12: while $(ones < h)$ do
13:     Read, starting at the $p$-th bit of *src*, $w$ consecutive bits from *src* and interpret them, assuming straight-
        forward binary representation, as a value $os \in \{0,...,n-1\}$.
14:     Read, starting at the $(p+w)$-th bit of *src*, $w$ consecutive bits from *src* and interpret them as a value
        $cn \in \{0,...,n-1\}$.
15:     Starting at the $os$-th bit of *rawres*, search through the bits of *rawres* for the $cn$-th appearance of the value
        0. (When having reached the end of *rawres* (the index $n-1$), continue at the begin of *rawres* (index 0)).
        Switch the bit with the $cn$-th appearance of 0 from 0 to 1.
16:     if (the index (in *rawres*) of the bit switched is $< z$) then
17:         $ones \leftarrow ones + 1$
18:     end if
19:     $p \leftarrow p + 2w$
20: end while
21: Return with the first $z$ bits of *rawres* as the result.
```

B.2 Runtime-optimized Hash with Predetermined Hamming Weight

The following algorithm is a variation of the first algorithm, runtime-optimized for cases in which the fixed Hamming weight is about half the bit length of the output. The idea is to first assign a conventional hash (which implies that each resulting bit has a statistical probability of 50% to be 1) and then "correct" the number of 1's.

Given the bit string $input \in \{0,1\}^*$ to be hashed, the bit length $z \in \mathbb{N}^*$ of the bit string to be produced, the required Hamming weight $h \in \{0, ..., z\}$, and the standard cryptographic hash function H of output bit length $hlen \in \mathbb{N}^*$, the algorithm performs the following steps:

```
 1: Determine the number of bits needed to represent the indices $(0, ..., z-1)$:
    $w \in \mathbb{N}^* \leftarrow \lceil \log_2 z \rceil$
 2: Determine the bit lengths of the raw (uncut) result bit string rawres:
    $n \in \mathbb{N}^* \leftarrow 2^w$

    /* Create rawres and assign it a "pseudo-random" initial value */
 3: for $(bnr \in \mathbb{N}^* \leftarrow 0,\ rawres \in \{0,1\}^* \leftarrow$ "",  $ims \in \{0,1\}^* \leftarrow input;\ bnr < n;\ bnr \leftarrow bnr + hlen)$ do
 4:     $hims \in \{0,1\}^{hlen} \leftarrow H(ims)$
 5:     Append hims to the raw (uncut) result bit string rawres:
        $rawres \leftarrow rawres \,||\, hims$
 6:     Create a fresh input string for the next call of $H$:
        $ims \leftarrow ims \,||\, hims$
 7: end for

 8: Determine rawres's current Hamming weight $hw \in \mathbb{N}^0$ by counting the 1's in the first $z$ bits of rawres.
 9: if $(hw = h)$ then
10:     Return with the first $z$ bits of rawres as the result.
11: end if

12: if $(hw < h)$ then
13:     Identify 0 as the bit value whose number of appearances must be reduced:
        $r \in \{0,1\} \leftarrow 0$
14:     Computes the number of superfluous 0-bits:
        $d \in \mathbb{N}^0 \leftarrow h - hw$
15: else
16:     Identify 1 as the bit value whose number of appearances must be reduced:
        $r \leftarrow 1$
17:     Computes the number of superfluous 1-bits:
        $d \leftarrow hw - h$
18: end if
19: Determine the maximum number of "pseudo-random" bits needed:
    $m \in \mathbb{N}^* \leftarrow (n - z + d) \cdot w \cdot 2$

    /* Produce a bit string src as a source of "pseudo-random" indices */
20: for $(bnr \leftarrow 0,\ src \in \{0,1\}^* \leftarrow$ "";  $bnr < m;\ bnr \leftarrow bnr + hlen)$ do
21:     $hims \in \{0,1\}^{hlen} \leftarrow H(ims)$
22:     Append hlen bits to the source of "pseudo-random" indices:
        $src \leftarrow src \,||\, hims$
23:     Create a fresh input string for the next call of $H$:
        $ims \leftarrow ims \,||\, hims$
24: end for

25: Initialize the pointer from where to read the next pair of "pseudo-random" indices from the bit string src:
    $p \in \mathbb{N}^0 \leftarrow 0$
26: while $(d > 0)$ do
27:     Read, starting at the $p$-th bit of src, $w$ consecutive bits from src and interpret them as a value $os \in$
        $\{0, ..., n-1\}$.
28:     Read, starting at the $(p + w)$-th bit of src, $w$ consecutive bits from src and interpret them as a value
        $cn \in \{0, ..., n-1\}$.
```

```
29:     Starting at the $os$-th bit of $rawres$, search through the bits of $rawres$ for the $cn$-th appearance of $r$.
        (When having reached the end of $rawres$, continue at the begin). Negate the value of the bit of $r$'s $cn$-th
        appearance.
30:     **if** (the index (in $rawres$) of the bit switched is $< z$) **then**
31:         $d \leftarrow d - 1$
32:     **end if**
33:     $p \leftarrow p + 2w$
34: **end while**
35: Return with the first $z$ bits of $rawres$ as the result.
```

Note that the two algorithms above are just a demonstration of feasibility, very simple, straight-forward constructions from an existing multi-purpose cryptographic hash function. We expect especially designed, more efficient algorithms for the generation of cryptographic hashes of a constant, predetermined Hamming weight to be developed.