Practical Affiliation-Hiding Authentication from Improved Polynomial Interpolation

(Full Version[⋆])

Mark Manulis and Bertram Poettering

Cryptographic Protocols Group
Department of Computer Science
TU Darmstadt & CASED
Germany
mark@manulis.eu , bertram.poettering@cased.de

Abstract. Among the plethora of privacy-friendly authentication techniques, affiliation-hiding (AH) protocols are valuable for their ability to hide not only identities of communicating users behind their affiliations (memberships to groups), but also these affiliations from non-members. These qualities become increasingly important in our highly computerized user-centric information society, where privacy is an elusive good.

Only little work on practical aspects of AH schemes, pursuing optimized implementations and deployment, has been done so far, and the main question a practitioner might ask — whether affiliation-hiding schemes are truly practical today — remained widely unanswered. Improving upon recent advances in the area of AH protocols, in particular on pioneering results in the *multi-affiliation* setting, we can give an affirmative answer to this question. To this end, we propose numerous algorithmic optimizations to a recent AH scheme leading to a remarkable performance gain. Our results are demonstrated not only at theoretical level, but we also offer implementations, performance measurements, and comparisons. At the same time, our improvements advance the area of efficient polynomial interpolation in finite fields, which is one of our building blocks.

Keywords. affiliation-hiding authentication, privacy-oriented cryptography, polynomial interpolation, IHME

1 Introduction and Background

Secure communication in modern computer networks is usually achieved by means of authentication mechanisms provided by public key infrastructures (PKI). However, these mechanisms, in particular those built from traditional signature schemes, cannot satisfy the increasing demand of users for better privacy and anonymity. A valid digital signature reveals per definition the identity (public key) of the signer, and serves as strong evidence for the authentication attempt of a particular PKI user. Achieving authentication while preserving privacy of users upon their interactions is a challenging task due to the seemingly contradictory nature of both requirements. Nevertheless, many cryptographic authentication techniques, both interactive and non-interactive, aiming at various flavors of privacy, have been proposed, and the area remains a focus of ongoing research.

[⋆] A preliminary version of this paper appears in ACM ASIACCS 2011. This is the full version.

1.1 Affiliation-Hiding Authentication

Affiliation-Hiding (AH) is a property of privacy-preserving authentication protocols such as secret handshakes [1,2,6,16,18,29,30,32], and, more generally, affiliation-hiding key establishment protocols [14, 15, 22]. These schemes hide user identities by means of group authentication, but unlike group signatures [3, 7], that also offer this property, the distinguishing privacy property of AH protocols is that they also hide the actual groups (affiliations) of participants: Whenever the affiliations of both participants match, the authentication is successful, whereas, in all other cases, the protocol terminates without leaking any information about the groups, except for the fact that they do not match. In this light, AH protocols are interactive and bear high potential for protecting privacy in online communications. Moreover, the intrinsic requirement of AH protocols is to support (efficient) revocation of group membership, which is usually performed by respective Group Authorities (GAs), who manage their own groups independently of each other. AH protocols may have many variations, which are highlighted, from the practical perspective, below.

(Un)Linkability AH protocols can be of two flavors: linkable or unlinkable. In linkable protocols, such as [2, 6, 15, 22, 30], users communicate using pseudonyms for which they hold corresponding group membership credentials. These pseudonyms can be revoked by the GAs, i.e. put into revocation lists, which are distributed in an authenticated manner. Linkable protocols are usually more efficient than unlinkable protocols. In unlinkable protocols [1, 16, 18, 29], in order to prevent any correlation amongst sessions of the same user, complex group management techniques are deployed or some security compromise is taken into account: For example, [1] does not support revocation, [16] needs synchronization of revocation epochs amongst members, [29] requires costly group signatures and broadcast encryption techniques, while [18] attempts to combine group signatures with verifier-local revocation and private conditional oblivious transfer schemes.

Trustworthiness of Group Authorities In all AH protocols, group membership credentials are issued to users by respective GAs. Yet, existing protocols differ with regard to the amount of trust put in these GAs. While the majority of protocols assumes that GAs are trusted with both security of management of the group and privacy for its members, some recent protocols [22, 23] aim at relaxing the latter trust relationship by extending privacy protection of users even against malicious GAs, sometimes with considerable extra costs such as zero-knowledge proofs in [22].

Multi-Group Affiliation-Hiding Authentication Protocols Most AH protocols are *single-group* protocols: they can only cope with one input membership credential per user and session. In practice this is a severe limitation, as, for example, in social networks, either distributed or centralized, where users become members of multiple, independent interest groups. In these multi-affiliation settings, the purpose of an AH protocol is to determine whether two communication participants have interest groups in common and which groups these are. This problem, also known as the *group discovery* problem [15], can be solved with a single-group protocol only by trying many different combinations of group credentials. This is inefficient since the number of required sessions is quadratic in the number of memberships per user.

Only few AH protocols [5, 17, 21], which we call *multi-group* protocols, can efficiently handle the group discovery problem, i.e. perform group discovery using multiple membership credentials within one protocol session. In these protocols, the overall computational workload (in terms of public-key operations) and the bandwidth complexity remain linear in the number of input credentials, in contrast to the naïve combinatorial approach. Yet, amongst these protocols, only the multi-group AH solution of [21] can be seen as a generic transformation of a single-group AH protocol into a multi-group one, and its concrete instantiation is so far the only one that captures multiple input credentials per user and session within an adequate security model and is supported by a formal security proof. The conceptual advantage of [21] is that their generic transformation, based on a new primitive, called *Index-Hiding Message Encoding (IHME)*, provably preserves properties of the single-group protocol to which IHME is applied, being these its (un)linkability or the amount of trust set into GAs. At a high level, IHME combines messages of n single-group AH protocols, each executed with a different group credential, into a structure of the same size, in such a way, that it is impossible for the communication partner to tell whether a credential of some particular group has been used to compute that structure, unless this partner has himself a valid credential for that group. The so-far only known realization of IHME (see [21]) is based on polynomial interpolation over finite fields.

1.2 How Practical is Affiliation-Hiding?

Prior work has advanced in designing new AH protocols and developing security models to address different security and privacy flavors. Nevertheless, only little is known about the practical deployability of these protocols, as their implementation aspects, not to mention their concrete performance, have not been analyzed in most cases. In fact, the few estimations given in the corresponding papers are of theoretical nature only, and refer to the number of required public-key operations, e.g. modular exponentiations. Although we know that public-key operations are most expensive, neglecting other costs in AH protocols may not be the right approach, as we discuss now.

Recall from the above discussion, that only multi-group AH protocols, such as [17,21], are of practical interest, and indeed, the computation costs they offer in terms of public-key operations are linear. Intuitively, this is also a lower bound, as a linear number of group membership certificates must be processed by each of the protocol participants. However, both existing protocols have hidden *quadratic* (!) costs, regarding either symmetric decryption as in [17], or finite field operations (e.g. multiplications/inversions) in [21]. Are these hidden costs? Or, in other words, is the asymptotically quadratic computational complexity of current multi-group AH protocols really negligible in practice, where users may become members of lots of groups?[1] How does the computation scale? And, finally, is affiliation-hiding really practical today?

1.3 Contributions and Organization

For the benefit of their general approach over the dedicated construction in [17], this paper gives an in-depth analysis of [21], aiming to answer the aforementioned questions, and comes to the result that affiliation-hiding techniques can be made truly practical.

To reach this goal, however, we suggest numerous optimizations at the algorithmic level to both the polynomial-based construction of IHME and the entire AH protocol from [21], resulting in remarkable improvements of efficiency in comparison to the original schemes. We demonstrate this not only through theoretical estimations but also with complete implementations and measurements. Along the paper, we provide various algorithms with regard to efficient polynomial interpolation, and even propose our own optimizations to this field. Furthermore, we provide a complete dissection of the multi-group AH protocol from [21], showing how to optimize all of its further building blocks (besides IHME), with the result that our protocol considerably outperforms its predecessor.

All algorithms that appear in this paper, are written in an implementation-close way, and can be readily translated into the program code.

[1] For instance, an average Facebook user is connected to about 80 community pages and groups [10].

Organization We start, in Section 2, by giving a basic overview over both IHME and the AH protocol from [21]. In Section 3 we focus on IHME and show how to realize it in practice: We show that some well-known interpolation algorithms do not serve for IHME implementations, and that although some less known algorithms can be used, they are not as efficient as those we introduce. In Sections 3.5 and 3.8 we further give concrete implementation results and analyze the performance of IHME according to various metrics. In Section 4 we focus on further building blocks and techniques that constitute (in addition to IHME) the entire AH protocol from [21]. Here, too, we use numerous tricks and present various optimization techniques with regard to different phases of the protocol, namely group setup, the registration of users to groups, and the actual multi-group handshake protocol. In Section 5 we give concrete implementation results and analyze the performance and scalability of the entire AH protocol.

We stress that our optimizations are performed at the algorithmic level and are of independent interest since e.g. polynomial interpolation over finite fields appears in many other cryptographic schemes whose practical deployment could also benefit from this work.

2 Initial Techniques: IHME and Multi-Group AH Protocols

In this section, we briefly recall the IHME technique and the multi-group AH scheme from [21], which serves as a starting point for our optimized construction.

2.1 Index-Hiding Message Encoding (IHME)

IHME is an encoding technique recently introduced and constructed in [21]. It is specified for some sets of *indices* $\mathcal{I}$ and *messages* $\mathcal{M}$, and consists of two algorithms iEncode and iDecode, such that $\mathcal{S} \leftarrow \mathsf{iEncode}(\mathcal{P})$ pools a set of input index/message pairs $\mathcal{P} = \{(i_1, m_1), \ldots, (i_n, m_n)\} \subseteq \mathcal{I} \times \mathcal{M}$ into a single data structure $\mathcal{S}$, whereas $m \leftarrow \mathsf{iDecode}(\mathcal{S}, i_j)$ recovers m from $\mathcal{S}$ for any $1 \leq j \leq n$, such that $m = m_j$. The intrinsic security property of IHME is *index-hiding*. It ensures that an attacker, who learns $\mathcal{S}$ and might even know some of the indices and corresponding messages, cannot identify any other indices used to encode messages in $\mathcal{S}$.

A multi-group AH protocol can be constructed from a single-group AH protocol in a generic way using IHME as follows: The first user, say U_1, computes the first messages m_j for each of her group memberships using the single-group protocol, then computes the IHME structure $\mathcal{S}$ from these messages m_j using corresponding public group parameters as indices (or, alternatively, hashes of public group keys), and finally sends $\mathcal{S}$ to U_2, who can correctly decode and answer messages in those groups for which U_2 also has group credentials. The IHME encoding is separately applied for each round of the original single-group protocol.

2.2 Multi-Group Affiliation-Hiding Authentication Protocol

The concrete multi-group AH protocol from [21] works, highly simplified, as follows. We remark that a more detailed explanation of the protocol will be given step by step in our discussions regarding the optimization of the protocol in Section 4.

Each GA creates its own group by independently selecting the corresponding public group RSA parameters (n, e, g), where n is the RSA modulus, e the public exponent, and g a generator of a large subgroup of $\mathbb{Z}_n^\times$ (CreateGroup algorithm). The GA keeps the corresponding secret exponent d as a private key to itself. Membership credential cred of a user with pseudonym id in a group with public key (n, e, g) and secret key d is $\mathsf{cred} = H_n(\mathsf{id})^d \bmod n$, i.e. the Full-Domain Hash RSA Signature on the pseudonym of the user, computed using an appropriate hash function H_n. This credential is granted to the user, typically over a secure channel, by the GA of that group during user registration ($\mathsf{AddUser}$) phase. The actual multi-group affiliation-hiding handshake protocol ($\mathsf{Handshake}$) takes two communication rounds. In the first round, users exchange, using the above IHME technique, values of the form $\theta = (g^t \mathsf{cred}) \bmod n$, for all groups they are member of, where t is some random exponent that will be also used for the computation of the session key in a fashion similar to the basic Diffie-Hellman approach, yet in subgroup $\langle g \rangle \subseteq \mathbb{Z}_n^\times$. In the second round of the protocol, participants exchange confirmation messages for this key. Through their verification users also learn the intersection of their affiliations (groups), and by using the established key they can continue communicating securely.

3 Optimized IHME from Improved Polynomial Interpolation

A possible instantiation of IHME, based on polynomial interpolation in finite fields, is proposed in [21]. In this section, we describe this construction in detail, give several optimized algorithms for its implementation, and compare them in respect to computational efficiency and memory consumption. Note that in [21] these implementational aspects were left unconsidered.

All algorithms presented below make computations in an arbitrary finite field $\mathbb{F}$. Efficient implementation of field arithmetic and element representation is a wide field of research and out of the scope of this paper; see Hankerson et al. [13, Chapter 2] for a comprehensive overview. Generally speaking, fields of small characteristic (e.g. $\mathbb{F} = GF(2^k)$ for some k) offer speed advantages on SIMD machines and dedicated hardware, while modern PCs with 32/64 bit ALUs benefit from fields of large characteristic (e.g. $\mathbb{F} = GF(p)$ for a large prime p).

In the analysis of the following algorithms, we measure computational performance by counting the number of expensive field operations, i.e. multiplications ($c \leftarrow ab$), inversions ($c \leftarrow a^{-1}$), and divisions ($c \leftarrow a/b$). As $a/b = ab^{-1}$, divisions can always be implemented at the cost of one inversion and one multiplication. However, often both operations can be conflated into a single operation of the cost of one inversion [13, Section 2.3.6]. In the following, we denote the time needed to perform a multiplication, an inversion, or a division, by M, I and D, respectively. In practice, it is reasonable to expect $I \approx D \approx 60M$ (cf. [13, Section 5.1.5]).

3.1 Polynomial-based IHME Construction

Let $\mathbb{F}$ denote an arbitrary finite field. After setting $\mathcal{I} = \mathcal{M} = \mathbb{F}$, an index-hiding message encoding scheme $\mathsf{IHME} = (\mathsf{iEncode}, \mathsf{iDecode})$ with index space $\mathcal{I}$ and message space $\mathcal{M}$ is constructed by the following algorithms:

$\mathsf{iEncode}(\mathcal{P})$: On input of $\mathcal{P} = \{(i_1, m_1), \ldots, (i_n, m_n)\} \subseteq \mathcal{I} \times \mathcal{M} = \mathbb{F}^2$, the encoding is defined as the list $\mathcal{S} = (c_{n-1}, \ldots, c_0)$ of coefficients of the polynomial $p(x) = \sum_{k=0}^{n-1} c_k x^k \in \mathbb{F}[x]$ that interpolates all points in $\mathcal{P}$, i.e. $p(i_j) = m_j$ for all $(i_j, m_j) \in \mathcal{P}$. Note that this polynomial exists uniquely (cf. Theorem 1), i.e. the $\mathsf{iEncode}$ algorithm is deterministic.

$\mathsf{iDecode}(\mathcal{S}, i)$: On input of $\mathcal{S} = (c_{n-1}, \ldots, c_0)$ and index $i \in \mathcal{I}$, this algorithm outputs the evaluation $m = p(i) = \sum_{k=0}^{n-1} c_k i^k$ of p at position i.

We note that efficient implementations of $\mathsf{iEncode}$ and $\mathsf{iDecode}$ are implied by efficient algorithms for polynomial interpolation and evaluation. Furthermore, we note that the index-hiding property of the above construction holds information-theoretically, as proven in [21].

3.2 Polynomial Interpolation[2]

The following well-known theorem [28] ensures existence and uniqueness of interpolation polynomials.

Theorem 1 (Polynomial Interpolation). *In a field $\mathbb{F}$, let $(x_1, y_1), \dots, (x_n, y_n) \in \mathbb{F} \times \mathbb{F}$ be n pairs of elements satisfying $i \neq j \Rightarrow x_i \neq x_j$. Then there exists a polynomial $p \in \mathbb{F}[x]$ of degree $\deg(p) < n$ that interpolates all points (x_i, y_i), i.e. $y_i = p(x_i)$ for all $1 \le i \le n$. Moreover, this polynomial exists uniquely.*

For fixed $n \in \mathbb{N}$, the set Π^n consisting of all polynomials $p \in \mathbb{F}[x]$ of degree $\deg(p) \le n$ naturally forms a vector space over $\mathbb{F}$. Algorithms for polynomial interpolation [28] usually represent computed polynomials in Π^n by the coefficients of the corresponding linear combination of some basis elements of Π^n. While the *monomial basis* $\{1, x, x^2, \dots, x^n\}$ seems to be the most versatile one, popular interpolation algorithms do not refer to it, but instead compute coefficients in respect to specially crafted bases, that often depend on the specific problem instance. We stress that such algorithms might not serve for secure IHME implementations. For example, the bases of two well-known interpolation algorithms, namely Lagrange and Newton Interpolation, are directly dependent on deployed x-abscissas. This behavior contradicts the desired index-hiding property of IHME, as x-values (i.e. indices) would have to be included in IHME structures $\mathcal{S}$.

Lagrange Interpolation In the terms of Theorem 1, a polynomial that interpolates $(x_1, y_1), \dots, (x_n, y_n)$ is given by

$$p(x) = \sum_{k=1}^{n} \Big(y_k \prod_{\substack{j=1 \\ j \neq k}}^{n} \frac{x - x_j}{x_k - x_j} \Big).$$

Correctness of this approach can be seen as follows: For a fixed k, function

$$L_k(x) = \prod_{j=1, j\neq k}^{n} \frac{x - x_j}{x_k - x_j}$$

is a polynomial of degree $n-1$ which evaluates to 1 at position x_k, and evaluates to 0 at positions x_l for all $1 \le l \le n$, $l \neq k$. It follows that $p(x) = \sum_{k=1}^{n} y_k L_k(x) \in \Pi^{n-1}$ interpolates $(x_1, y_1), \dots, (x_n, y_n)$.

Relating this to the said above, Lagrange's method for polynomial interpolation *does not* look for coefficients of a linear combination of fixed basis elements of Π^{n-1}, but rather outputs vectors $L_1, \dots, L_n$ in Π^{n-1} such that p is their 'trivial' linear combination with coefficients y_i.

In practice, Lagrange's method is rarely used for polynomial interpolation for being awkward and inefficient. Its importance is more on the theoretical side, e.g. for proving 'existence' in Theorem 1.

Newton Interpolation A far more efficient (and popular) way to perform polynomial interpolation in the context of scientific computing is due to Newton. We refer to [28, Section 4.2] for a detailed exposition, but stress that this method outputs coefficients a_k in respect to *Newton*

[2] We clarify that by 'polynomial interpolation' we comprehend the determination of a set of coefficients that fully describe the sought for polynomial. In the literature, however, often the evaluation of this polynomial at given points is subsumed under the same term, possibly without explicit computation of the coefficients.

bases $\{N_1, \ldots, N_n\}$ of Π^n, which are instance-specific as well. In particular, they consist of the elements

$$N_k(x) = \prod_{j=1}^{k-1} (x - x_j).$$

Corresponding coefficients $a_k = [y_1, \ldots, y_k]$ can be efficiently computed via *divided differences* [28], to obtain the interpolating polynomial as $p(x) = \sum_{k=1}^{n} a_k N_k(x)$.

3.3 Interpolation without Precomputation

An algorithm for polynomial interpolation that outputs coefficients in respect to monomial basis $\{1, x, x^2, \ldots, x^n\}$ of Π^n is due to Björck and Pereyra [4], and portrayed below as Algorithm 1. It has (quadratic) running time

$$\frac{n(n-1)}{2}(D + M)$$

and, as most of the algorithms proposed in this section, needs no extra storage, as all calculations can be implemented 'in place'.

Algorithm 1 Polynomial Interpolation

```
Input: Pairs $(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{F} \times \mathbb{F}$ with $i \neq j \Rightarrow x_i \neq x_j$
Output: Coefficients $c_0, \ldots, c_{n-1} \in \mathbb{F}$ such that $y_i = \sum_{k=0}^{n-1} c_k (x_i)^k \ \forall i$
  $(c_0, \ldots, c_{n-1}) \leftarrow (y_1, \ldots, y_n)$
  for $k = 0$ to $n - 2$ do
    for $j = n - 1$ downto $k + 1$ do
      $c_j \leftarrow (c_j - c_{j-1})/(x_{j+1} - x_{j-k})$
    end for
  end for
  for $k = n - 2$ downto $0$ do
    for $j = k$ to $n - 2$ do
      $c_j \leftarrow c_j - x_{k+1} c_{j+1}$
    end for
  end for
```

Algorithm 1 already solves the problem of polynomial interpolation in reasonable time. However, we can further improve computational efficiency by reducing the number of divisions from $O(n^2)$ to 1, while at the same time moderately increasing the number of multiplications.

The trick is to represent intermediate variables c not as field elements, but as *fractions*

$$c/d \,\hat{=}\, (c, d) \in \mathbb{F} \times \mathbb{F}^{\times},$$

where we identify fraction $(c, 1)$ with field element c.

Two fractions $(c, d), (c', d') \in \mathbb{F} \times \mathbb{F}^{\times}$ are considered *equivalent* (or equal) if $cd' = c'd$. Fractions are normalized to equivalent field elements by the *reduction mapping* $(c, d) \mapsto cd^{-1}$. If n of these reductions are to be computed in batch, the required n divisions can be conflated into a single inversion, at the cost of some additional multiplications (see Algorithm 8 in Appendix B, or Cohen [8, Algorithm 10.3.4] and Montgomery [25]).

The benefit achieved by the redundancy introduced by the 'computing with fractions' technique is that most divisions can be replaced by multiplications, as the example $(c, d)/(c', d') =$

(cd', dc') illustrates. This technique, applied to Algorithm 1, results in Algorithm 2, which has computational performance

$$\Big(\frac{5n(n-1)}{2}+1\Big)M + 1I.$$

Its speed advantage over Algorithm 1 is obvious for $D \gg M$. We note that Algorithm 2 needs extra storage for $n-1$ auxiliary variables $d_1, \ldots, d_{n-1}$.

Algorithm 2 Interpolation with Deferred Inversion

```
Input: Pairs $(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{F} \times \mathbb{F}$ with $i \neq j \Rightarrow x_i \neq x_j$
Output: Coefficients $c_0, \ldots, c_{n-1} \in \mathbb{F}$ such that $y_i = \sum_{k=0}^{n-1} c_k (x_i)^k \; \forall i$
  $(c_0, \ldots, c_{n-1}) \leftarrow (y_1, \ldots, y_n)$
  for $j = n-1$ downto $1$ do
    $c_j \leftarrow c_j - c_{j-1}$
    $d_j \leftarrow x_{j+1} - x_j$
  end for
  for $k = 1$ to $n-2$ do
    for $j = n-1$ downto $k+1$ do
      $c_j \leftarrow c_j d_{j-1} - c_{j-1} d_j$
      $d_j \leftarrow d_j d_{j-1} (x_{j+1} - x_{j-k})$
    end for
  end for
  $c_j \leftarrow c_j d_j^{-1}$ for all $1 \leq j \leq n-1$        (see note on batched reduction)
  for $k = n-2$ downto $0$ do
    for $j = k$ to $n-2$ do
      $c_j \leftarrow c_j - x_{k+1} c_{j+1}$
    end for
  end for
```

3.4 Interpolation with Precomputation

In some occasions polynomial interpolations have to be computed many times in succession, with fixed inputs x_i but variable inputs y_i. These cases are susceptive for improvements in efficiency by splitting calculations into a precomputation phase (on input the x_i), and a computation phase (on input the y_i, plus some precomputed state). The overall costs of polynomial interpolation are then determined by the costs of the second step, which might be more efficient than a regular interpolation (Algorithm 1 and 2).

Observe that for the coefficients c_k of a polynomial $p(x) = \sum_{k=0}^{n-1} c_k x^k \in \mathbb{F}[x]$ that passes through a set of points $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ the following system of equations holds:

$$\begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ \vdots & & & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} \end{pmatrix} \begin{pmatrix} c_0 \\ \vdots \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \tag{1}$$

The $(n \times n)$-matrix $V = V(x_1, \ldots, x_n)$ on the left is called *Vandermonde matrix* [28]. It is known from Numerical Analysis that V is invertible iff $i \neq j \Rightarrow x_i \neq x_j$. After computing V^{-1} in a precomputation step, one can solve (1) for $c_0, \ldots, c_{n-1}$ by calculating $(c_0, \ldots, c_{n-1})^T = V^{-1}(y_1, \ldots, y_n)^T$, essentially performing a matrix/vector multiplication with $n^2 M$ costs (cf. Algorithm 3). Explicit formulae for V^{-1} are developed in [9] and included as Algorithm 7 in Appendix A.

Algorithm 3 Interpolation after Precomputation

```
Input: Matrix $V^{-1} = (m_{1,1}, \dots, m_{n,n}) \in \mathbb{F}^{n \times n}$ as output by Algorithm 7, elements $y_1, \dots, y_n \in \mathbb{F}$
Output: Coefficients $c_0, \dots, c_{n-1} \in \mathbb{F}$ such that $y_i = \sum_{k=0}^{n-1} c_k (x_i)^k \ \forall i$
  for $i = 1$ to $n$ do
    $c_{i-1} \leftarrow 0$
    for $j = 1$ to $n$ do
      $c_{i-1} \leftarrow c_{i-1} + m_{i,j} y_j$
    end for
  end for
```

3.5 Performance Comparison of Interpolation Algorithms

In Figure 1, we compare presented Algorithms 1, 2 and 3 in efficiency. It becomes obvious that Algorithm 1 from [4] is actually not competitive with our optimized variant (Algorithm 2), and that precomputations can, moreover, roughly halve running time. Time consumption, on the right axis, is estimated by assuming $M = 0.44\mu s$, as measured in our test implementation (see also notes in Figure 4).

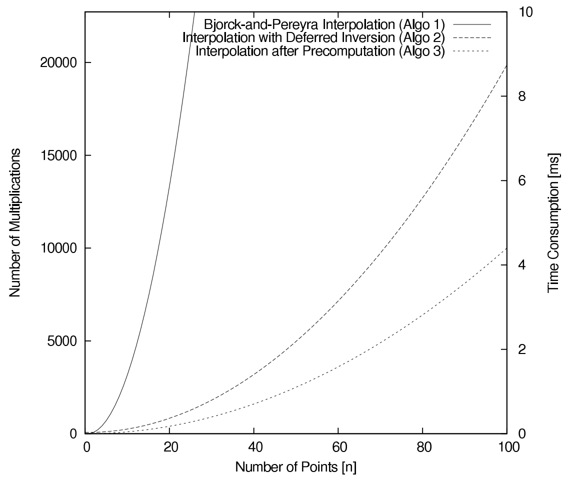

Fig. 1. Efficiency comparison of interpolation algorithms 1, 2 and 3. The axis on the left reflects the number of executed field multiplications (we assume $D = I = 60M$), the axis on the right indicates time consumption for a finite field $\mathbb{F}$ of about 2^{80} elements.

3.6 Polynomial Evaluation

For a given set $c_0, \dots, c_{n-1} \in \mathbb{F}$ of coefficients, the naïve way of evaluating polynomial $p(x) = \sum_{k=0}^{n-1} c_k x^k \in \mathbb{F}[x]$ at a given point $x \in \mathbb{F}$ would have $O(n^2)$ performance. Deployment of Horner's Scheme (Algorithm 4), however, reduces the running time to $(n-1)M$.

3.7 Interleaved IHME

In Sections 3.3, 3.4 and 3.6 we have seen implementations of IHME's iEncode and iDecode routines of performance $O(n^2)$ and $O(n)$, respectively. However, their running time was measured with

Algorithm 4 Polynomial Evaluation

```
Input: Coefficients $c_0, \ldots, c_{n-1} \in \mathbb{F}$ and $x \in \mathbb{F}$
Output: Element $y \in \mathbb{F}$ with $y = \sum_{k=0}^{n-1} c_k x^k$
  $y \leftarrow c_{n-1}$
  for $k = n-2$ downto $0$ do
    $y \leftarrow c_k + xy$
  end for
```

regards to a fixed finite field $\mathbb{F}$. In practice [21], these fields may become rather large, e.g. $|\mathbb{F}| \geq 2^{1024}$, and IHME will perform accordingly slow (although still in $O(n^2)$). In this section, we present an *interleaving technique* which allows to further speedup IHME. Note that the algorithms remain in $O(n^2)$, it is rather the constant that is considerably reduced.

Consider, for instance, an IHME setting with $\mathbb{F} = GF(2^{1024})$ and $\mathcal{M} = \mathcal{I} = \mathbb{F} \cong \{0,1\}^{1024}$. Instead of encoding messages $m_1, m_2, \ldots \in \mathcal{M}$ over this field, one could split all messages m_i into, say, 8 chunks $m_{i,1}, \ldots, m_{i,8}$, each of length $1024/8 = 128$. Now, using IHME over field $\mathbb{F}' = GF(2^{128})$, all $m_{i,1}$ can be IHME-encoded into a structure $\mathcal{S}_1$, all $m_{i,2}$ can be independently encoded into a structure $\mathcal{S}_2$, and so on. The overall encoding is then $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_8)$. The gain in efficiency is caused by the trade of superlinear costs of finite field arithmetics for linear costs of IHME interleaving. We formalize the ideas of this paragraph as follows.

ν-fold IHME For an arbitrary finite field $\mathbb{F}$ and $\nu \in \mathbb{N}$, after setting $\mathcal{I} = \mathbb{F}$ and $\mathcal{M} = \mathbb{F}^\nu$, an index-hiding message encoding scheme IHME = (iEncode, iDecode) with index space $\mathcal{I}$ and message space $\mathcal{M}$ is constructed from standard IHME′ = (iEncode′, iDecode′) over $\mathbb{F}$ as follows (cf. Section 3.1):

iEncode($\mathcal{P}$) : On input of $\mathcal{P} = \{(i_1, (m_{1,1}, \ldots, m_{1,\nu})), \ldots, (i_n, (m_{n,1}, \ldots, m_{n,\nu}))\} \subseteq \mathcal{I} \times \mathcal{M} = \mathbb{F} \times \mathbb{F}^\nu$, the encoding is defined as the list $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_\nu)$ of IHME′-encodings

$$\mathcal{S}_k = \mathsf{iEncode}'(\{(i_j, m_{j,k})\}_{1 \leq j \leq n}), \text{ for } 1 \leq k \leq \nu.$$

iDecode($\mathcal{S}, i$) : On input of $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_\nu)$ and index $i \in \mathcal{I}$, this algorithm outputs $m = (m_1, \ldots, m_\nu)$ where

$$m_k = \mathsf{iDecode}'(\mathcal{S}_k, i), \text{ for } 1 \leq k \leq \nu.$$

In the following sections, by iEncode($\mathcal{P}, \Pi, \nu$), for prime Π and natural number ν, we denote an IHME-encoding with index space $\mathcal{I} = [0, \Pi - 1]$ and message space $\mathcal{M} = [0, \Pi^\nu - 1]$, i.e. $\mathcal{P} \subseteq \mathcal{I} \times \mathcal{M}$. Such an IHME-scheme is constructed by exploiting the existence of finite field $\mathbb{F} = GF(\Pi)$ and the natural and efficient bijections $[0, \Pi - 1] \to \mathbb{F}$ and $[0, \Pi^\nu - 1] \to \mathbb{F}^\nu$ (e.g., for the latter, the representation to basis Π, i.e. $a \mapsto (a_0, \ldots, a_{\nu-1})$ where $a = \sum_{k=0}^{\nu-1} a_k \Pi^k$). Analogously, by iDecode($\mathcal{S}, \Pi, \nu, i$) we denote the corresponding IHME-decoding at index $i \in \mathcal{I}$.

3.8 Performance Gain with Interleaved IHME

The gain in efficiency over Standard IHME, achieved by deployment of ν-fold IHME, is illustrated in Figure 2. We use 1104 bit fields in the test case because fields of this size naturally emerge in the setting of affiliation-hiding protocols (cf. Sections 2.2 and 4). We observe that ν-fold IHME outperforms standard IHME by about 30%, for both underlying Algorithms 2 and 3.

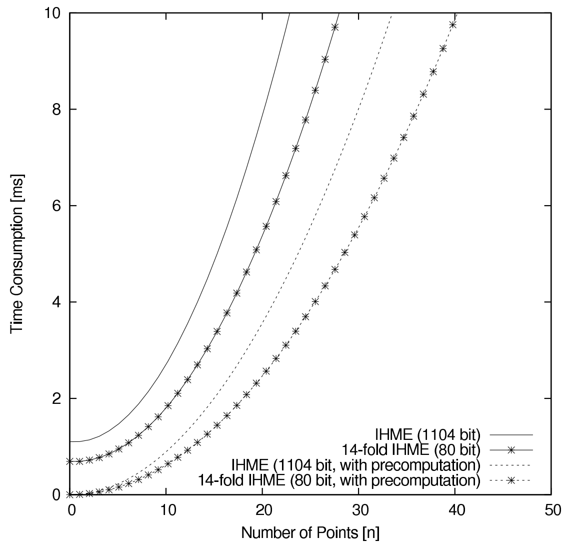

Fig. 2. Efficiency comparison of Standard IHME (Section 3.1) and ν-fold IHME (Section 3.7). Precisely, we compare Standard IHME over a 1104 bit field with (the more or less equivalent) 14-fold IHME over a 80 bit field (note that $80 \cdot 14 = 1120$), both without and with precomputations (Algorithms 2 and 3, respectively). The offset on the y-axis for interpolations without precomputation is due to the (relatively high) cost of the inversion in Algorithm 2.

4 Optimized Multi-Group Affiliation-Hiding Authentication Protocol

Our further optimizations towards truly practical affiliation-hiding authentication concern the different phases of the multi-group AH protocol from [21]. Along the description of its three main phases, namely the initialization of groups, registration of users, and executions of group-discovering handshakes, we introduce modifications that achieve substantial efficiency improvements, considering both computational performance and bandwidth consumption.

In all algorithms and protocols, by κ and κ' we denote security parameters of symmetric and asymmetric building blocks, respectively. Reasonable choices for κ, κ', hash functions and other building blocks will be proposed in Section 5.

4.1 Group Initialization Phase

The CreateGroup algorithm is run once per group authority and, hence, computational efficiency is not too important for its implementation. Instead, we decided to improve on storage size of group parameters. Recall that public parameters in [21] are triples (n, e, g), where n is a safe RSA modulus, e is an RSA exponent suitable for modulus n, and g is a generator of a maximum order subgroup of $\mathbb{Z}_n^\times$ such that $-1 \notin \langle g \rangle$. Our improved algorithm, shown as Algorithm 5, outputs RSA moduli n that are crafted in such a way that $g = 2$ and $e = 3$ are valid group generators and RSA exponents, respectively. As consequence, it suffices to store just modulus n as public group key.

In addition, in Section 4.3, we will see that choosing $g = 2$ offers an attractive opportunity of implementing the exponentiations in the Handshake protocol very efficiently.

We start by giving some fundamental number-theoretical definitions and relations. We refer to [24] for a more detailed exposition. A *safe prime* p is a prime number such that $p = 2p' + 1$, where p' is prime as well. For a safe prime p, the multiplicative group $\mathbb{Z}_p^\times$ has order $2p'$, and each of its subgroups has order 1, 2, p' or $2p'$ (by Lagrange's theorem).

The subgroup of order p' consists exactly of all squares in $\mathbb{Z}_p^\times$, it is hence called the subgroup of *quadratic residues* $\bmod p$, $QR(p)$ for short. Note that $QR(p)$ is generated by each square in $\mathbb{Z}_p^\times$, except by 1. The *Legendre symbol* $\left(\frac{\cdot}{p}\right) : \mathbb{Z}_p^\times \to \{-1, 1\}$ is defined by $\left(\frac{a}{p}\right) = 1 :\Leftrightarrow a \in QR(p)$.

The *Chinese Remainder Theorem* (CRT) states that $\mathbb{Z}_{pq} \cong \mathbb{Z}_p \times \mathbb{Z}_q$ for all (safe) primes p, q, implying also $\mathbb{Z}_{pq}^\times \cong \mathbb{Z}_p^\times \times \mathbb{Z}_q^\times$. The corresponding isomorphism is given by $a_n \mapsto (a_n \bmod p, a_n \bmod q)$, and its inverse by $(a_p, a_q) \mapsto a_p + ph$ for $h = (a_q - a_p)/p \pmod q$.

For all $m \in \mathbb{N}$, *Carmichael function* $\lambda(m)$ indicates the order of the largest cyclic subgroup in $\mathbb{Z}_m^\times$. For primes p we have $\lambda(p) = p - 1$, for RSA moduli $n = pq$ it holds that $\lambda(n) = \operatorname{lcm}(p - 1, q - 1)$. In particular, for *safe RSA moduli*, i.e. moduli $n = pq$ with safe primes $p = 2p' + 1, q = 2q' + 1$, we have $\lambda(n) = 2p'q'$.

Lemma 1. *Let $p = 2p' + 1$ be a safe prime. Then $p = 11 \pmod{12}$.*

Proof. As p' is a prime number we have $p' \in \{1, 5\} \pmod 6$. Case $p' = 1 \pmod 6$ leads to $p = 2p' + 1 = 3 \pmod{12}$, a contradiction. Hence $p' = 5 \pmod 6$ and $p = 11 \pmod{12}$. □

Lemma 2. *Let $n = pq$ for safe primes $p = 2p' + 1, q = 2q' + 1$ with $p = 3 \pmod 8$ and $q = 7 \pmod 8$. Then $g = 2$ generates a subgroup of $\mathbb{Z}_n^\times$ of maximal order. In addition, we have $-1 \notin \langle g \rangle$.*

Proof. It is known [24] that $\left(\frac{2}{p}\right) = -1$ and $\left(\frac{2}{q}\right) = 1$. Hence $g \in \mathbb{Z}_n$, considered as element $g \in \mathbb{Z}_p$, generates $\langle g \rangle_p = \mathbb{Z}_p^\times$, while $g \in \mathbb{Z}_q$ generates $\langle g \rangle_q = QR(q)$. By applying CRT, we see $\langle g \rangle_n \cong \langle g \rangle_p \times \langle g \rangle_q$ and that the order of $g \in \mathbb{Z}_n$ is $\operatorname{lcm}(2p', q') = 2p'q' = \lambda(n)$. In addition, as $q = 3 \pmod 4$, we have $\left(\frac{-1}{q}\right) = -1$, i.e. $-1 \notin QR(q) = \langle g \rangle_q$, and hence $-1 \notin \langle g \rangle_n$. □

Algorithm 5 outputs safe RSA moduli n and private keys d such that $g = 2$ is a generator of a maximum order subgroup of $\mathbb{Z}_n^\times$ with $-1 \notin \langle g \rangle$, and d and $e = 3$ are valid RSA signing and verification keys, respectively, i.e. $ed = 1 \pmod{\lambda(n)}$. Factor $p = 2p' + 1$ of $n = pq$ is chosen such that $p = 11 \pmod{12}$ and $p = 3 \pmod 8$, and hence $p = 11 \pmod{24}$, while for $q = 2q' + 1$ we require $q = 11 \pmod{12}$ and $q = 7 \pmod 8$, i.e. $q = 23 \pmod{24}$ (compare to Lemmas 1 and 2). The search for safe primes for which these congruences hold is performed by the two while loops. The remaining part of the algorithm is straight forward[3]. Note that $\lambda(n) = 2p'q'$, and hence $e = 3$ is always invertible $\bmod \lambda(n)$.

Further Improvements A speed improvement for Algorithm 5 is gained if the two primality tests in the condition of the first while loop are executed in an interleaved way. Fully establishing p's primality if small factors of $(p - 1)/2$ are easily detected is waste of computing power. Instead, a seek for small divisors in both p and $(p - 1)/2$ should be completed before starting any (expensive) Miller-Rabin exponentiation [24]. The analogue applies to the second while loop.

A further technique for improving computational efficiency of Algorithm 5 bases on ideas of Naccache[4]. In [26] he describes an algorithm for accelerated generation of safe primes: First,

[3] By $\mathsf{RandNum}([a, b])$ we denote the uniformly random choice of an integer x with $a \le x \le b$. By $\mathsf{IsPrime}(x)$ we denote the application of a (probabilistic) primality test, e.g. the Miller-Rabin test, to integer x.

[4] We point out that Naccache's original algorithm [26] is not optimal in the sense that it does not regard the result of Lemma 1: When seeking for prime candidates, instead of narrowing the focus on integers p with $p = 11 \pmod{12}$, his algorithm tries all p with $p = 3 \pmod 4$. Hence, in two of three cases neither $(p - 1)/2$ nor $2p + 1$ can be prime.

Algorithm 5 Implementing CreateGroup

```
Input: Security parameter $\kappa'$ (typically $1024 \leq \kappa' \leq 2048$)
Output: Public parameters $(n, e, g)$ and private key $d$, where $e = 3$ and $g = 2$
  $\ell \leftarrow \lfloor \kappa'/2 \rfloor$
  $p \leftarrow 24 \cdot \lfloor \mathsf{RandNum}([2^{\ell-1}, 2^{\ell} - 1])/24 \rfloor + 11$
  while $\neg\mathsf{IsPrime}(p) \vee \neg\mathsf{IsPrime}((p-1)/2)$ do
    $p \leftarrow p + 24$
  end while
  $q \leftarrow 24 \cdot \lfloor \mathsf{RandNum}([2^{\ell-1}, 2^{\ell} - 1])/24 \rfloor + 23$
  while $\neg\mathsf{IsPrime}(q) \vee \neg\mathsf{IsPrime}((q-1)/2)$ do
    $q \leftarrow q + 24$
  end while
  $\lambda = (p-1)(q-1)/2$
  $(n, e, g) \leftarrow (pq, 3, 2)$
  $d \leftarrow e^{-1} \bmod \lambda$
```

random integers $p = 2p' + 1$ satisfying $p = 11 \pmod{12}$ are drawn and tested for primality, until a prime is found. If, in addition, p' is prime then we are done. If, on the other hand, p' is not prime, then $P := 2p + 1$ is considered a candidate for being safe prime and is checked via a single invocation of IsPrime. All in all, fewer primality tests have to be run for finding safe primes. This technique can be integrated into both while loops of Algorithm 5, but attention is needed to ensure that the length of n stays in an acceptable range, and that p and q fulfill their specific congruences $\bmod 8$.

4.2 User Registration Phase

In the AddUser protocol from [21], users obtain credentials cred of the form $H_n(\mathsf{id})^d \bmod n$, i.e. Full-Domain Hash RSA signatures on their respective identities id. (A concrete instantiation of hash function $H_n : \{0,1\}^* \rightarrow \mathbb{Z}_n$ will be proposed in Section 5). We observe that, in the actual handshake protocol from [21], term $H_n(\mathsf{id})$ only occurs in contexts where values are *divided by* $H_n(\mathsf{id})$. For the sake of efficiency, and without influencing the scheme's security, we move these necessary inversions into the registration process, by altering the generation of user credentials to

$$\mathsf{cred} = (n, \sigma) \qquad \text{with} \qquad \sigma = H_n(\mathsf{id})^{-d} \bmod n.$$

A standard trick [19] to speed up private RSA operations is to apply CRT before computing exponentiations by d. More concretely, if the factorization of RSA modulus $n = pq$ is known, then $y = x^d \bmod n$ can be computed by CRT-decomposing x into $x_p = x \bmod p$ and $x_q = x \bmod q$, by computing $y_p = {x_p}^d = x_p^{d \bmod \varphi(p)} \pmod p$ and $y_q = {x_q}^d = x_q^{d \bmod \varphi(q)} \pmod q$, and by mapping (y_p, y_q) back to $\mathbb{Z}_n$, by applying CRT a second time. Besides the fact that exponentiations $\bmod p$ and $\bmod q$ can be computed substantially faster than exponentiations $\bmod n$, many intermediate values of this alternative signing method can be precomputed. An implementation of the AddUser algorithm that includes these optimizations is given in Algorithm 6.

4.3 Multi-Group Handshake Protocol

Users that possess a set $\mathcal{C} = \{\mathsf{cred}_1, \ldots, \mathsf{cred}_m\}$ of group credentials for their pseudonym id are ready to perform a multi-group Handshake session. Our optimized version of the protocol is presented in Figure 3. Note that by $H : \{0,1\}^* \rightarrow \{0,1\}^\kappa$ we denote a hash function.

Algorithm 6 Implementing $\mathsf{AddUser}$

```
Input: Public parameters $(n, e, g)$, private key $(p, q)$, pseudonym $\mathsf{id} \in \{0,1\}^*$
Output: User credential $\mathsf{cred} = (n, \sigma)$
Precompute: $d_p \leftarrow -e^{-1} \pmod{p-1}$
            $d_q \leftarrow -e^{-1} \pmod{q-1}$
            $u \leftarrow p^{-1} \pmod{q}$
  $h \leftarrow H_n(\mathsf{id})$
  $(h_p, h_q) \leftarrow (h \bmod p, h \bmod q)$
  $(\sigma_p, \sigma_q) \leftarrow ({h_p}^{d_p} \bmod p, {h_q}^{d_q} \bmod q)$
  $a \leftarrow u(\sigma_q - \sigma_p) \pmod{q}$
  $\sigma \leftarrow \sigma_p + pa$
  $\mathsf{cred} \leftarrow (n, \sigma)$
```

The principal enhancement of our design over [21] is the deployment of the more efficient Interleaved IHME technique presented in Section 3.7. In the original protocol [21], all messages exchanged in the two communication rounds are, when IHME encoded, considered as elements of a certain finite field $\mathbb{F}$. In particular, in the first round, padded RSA values of length $\kappa' + \kappa$ are exchanged. Hence, $\mathbb{F}$ has to be chosen accordingly large ($|\mathbb{F}| \geq 2^{1104}$ at least, see Section 5) and field arithmetics perform rather slow. In contrast, in our protocol (Figure 3), first-round messages $\theta \in [\,0, \varPi^{\nu} - 1\,]$ are encoded over a (much smaller) field of $\varPi$ elements, where $\varPi$ is a prime number slightly greater than 2^{κ}, using the ν-fold interleaved technique. Note that careful choice of $\varPi$, e.g. of low Hamming weight, allows impressively fast implementations of field arithmetics [13, Section 2.2.6]. Considering the second round messages, in [21], the per-group key confirmation messages are of length $\kappa' + \kappa$, where κ bits would suffice. In our protocol, however, confirmation messages are shortened to this more reasonable level and encoded using IHME, again over the field of $\varPi \approx 2^{\kappa}$ elements. Both these optimizations lead to a considerable boost of computational and bandwidth efficiency, when compared to a naïve implementation of [21].

A consequence of the switch to smaller fields is that also deployed IHME indices have to be chosen from a smaller set (see Section 3.1). While in [21] public RSA moduli n serve directly as group index, in our protocol the set of possible indices is reduced to the elements of $\mathbb{F}$, which is much too small for allowing a direct embedding of moduli n. We solve this problem by hashing public group parameters into $\mathbb{F}$. These hash values, $h_n = H(n)$, can further on be considered as convenient 'handles' for groups, and are therefore designated as the elements of the 'shared group' set groups that is output of the protocol (see line 31).

A further improvement over [21] is the more straight forward way of session key derivation. In [21], the key is computed as the hash value of a string that is composed of several per-group secrets. To achieve correctness of the protocol, both protocol participants have to mount this string in the same order. For this, a canonical ordering of groups has to be assumed. In our protocol (Figure 3), however, the key is computed as XOR-sum of per-group secrets (line 32), which, without loss of security, now can be computed in any order.

In contrast to [21], in our protocol, ephemeral exponents t (see lines 6, 7 and 19) are not chosen from $\mathbb{Z}_{n/2}$ (where n is an RSA modulus of length κ'), but from much smaller range $[\,0, 2^{2\kappa} - 1\,]$. This, again, leads to a notable gain in efficiency in the modular exponentiations. Under the common assumption [11, 12] that Discrete Logarithm Problem (DLP) in $\mathbb{Z}_n^{\times}$ is hard even when exponents are short, distributions of ephemeral keys with short and long exponents, respectively, are computationally indistinguishable from each other (see Lemma 3.6 in [12]). Hence, shortening ephemeral keys in the described way does not result in a considerably weaker security of the protocol.

```
    INITIATOR ON INPUTS $(\mathsf{id}, \mathcal{C})$

1   $\nu \leftarrow \lceil(\kappa' + \kappa)/\kappa\rceil$
2   $\mathcal{P}_1 \leftarrow \emptyset,\ \mathcal{T} \leftarrow \emptyset$
3   FOR ALL $\mathsf{cred} \in \mathcal{C}$:
4       PARSE $\mathsf{cred}$ AS $(n, \sigma)$
5       $h_n \leftarrow H(n)$
6       $(b, t) \leftarrow_R [0, 1] \times [0, 2^{2\kappa} - 1]$
7       $\vartheta \leftarrow (-1)^b 2^t \sigma \bmod n$
8       $k \leftarrow_R [0, \lfloor \Pi^\nu / n \rfloor - 1]$
9       $\theta \leftarrow \vartheta + kn$
10      $\mathcal{P}_1 \leftarrow \mathcal{P}_1 \cup \{(h_n, \theta)\}$
11      $\mathcal{T} \leftarrow \mathcal{T} \cup \{(n, h_n, t)\}$
12  $\mathcal{S}_1 \leftarrow \mathsf{iEncode}(\mathcal{P}_1, \Pi, \nu)$         $\xrightarrow{m\ =\ (\mathsf{id}, \mathcal{S}_1)}$
13                                                                  $\xleftarrow{m'\ =\ (\mathsf{id}', \mathcal{S}_1')}$
14  $\mathsf{sid} \leftarrow H(m \,\|\, m')$
15  $\mathcal{P}_2 \leftarrow \emptyset,\ \mathcal{R} \leftarrow \emptyset$
16  FOR ALL $(n, h_n, t) \in \mathcal{T}$:
17      IF $\mathsf{id}' \notin \mathsf{prl}$:
18          $\theta \leftarrow \mathsf{iDecode}(\mathcal{S}_1', \Pi, \nu, h_n)$
19          $r \leftarrow (\theta^3 H_n(\mathsf{id}'))^{2t} \bmod n$
20          $c_0 \leftarrow H(\mathsf{sid} \,\|\, r \,\|\, 0)$
21          $c_1 \leftarrow H(\mathsf{sid} \,\|\, r \,\|\, 1)$
22          $c_2 \leftarrow H(\mathsf{sid} \,\|\, r \,\|\, 2)$
23          $\mathcal{R} \leftarrow \mathcal{R} \cup \{(h_n, c_0, c_2)\}$
24      ELSE: $c_1 \leftarrow_R \{0, 1\}^\kappa$
25      $\mathcal{P}_2 \leftarrow \mathcal{P}_2 \cup \{(h_n, c_1)\}$
26  $\mathcal{S}_2 \leftarrow \mathsf{iEncode}(\mathcal{P}_2, \Pi, 1)$                  $\xleftarrow{\mathcal{S}_2'}$
27                                                                  $\xrightarrow{\mathcal{S}_2}$
28  $K \leftarrow 0^\kappa$, $\mathsf{groups} \leftarrow \emptyset$
29  FOR ALL $(h_n, c_0, c_2) \in \mathcal{R}$:
30      IF $\mathsf{iDecode}(\mathcal{S}_2', \Pi, 1, h_n) = c_2$:
31          $\mathsf{groups} \leftarrow \mathsf{groups} \cup \{h_n\}$
32          $K \leftarrow K \oplus c_0$
33
34  IF $\mathsf{groups} \neq \emptyset$ THEN
35      $(\mathsf{key}, \mathsf{partner}) \leftarrow (K, \mathsf{id}')$
36      TERMINATE WITH "ACCEPT"
37  ELSE
38      $(\mathsf{key}, \mathsf{partner}) \leftarrow (\bot, \bot)$
39      TERMINATE WITH "REJECT"
```

```
    RESPONDER ON INPUTS $(\mathsf{id}', \mathcal{C}')$

1   $\nu \leftarrow \lceil(\kappa' + \kappa)/\kappa\rceil$
2   $\mathcal{P}_1 \leftarrow \emptyset,\ \mathcal{T} \leftarrow \emptyset$
3   FOR ALL $\mathsf{cred} \in \mathcal{C}'$:
4       PARSE $\mathsf{cred}$ AS $(n, \sigma)$
5       $h_n \leftarrow H(n)$
6       $(b, t) \leftarrow_R [0, 1] \times [0, 2^{2\kappa} - 1]$
7       $\vartheta \leftarrow (-1)^b 2^t \sigma \bmod n$
8       $k \leftarrow_R [0, \lfloor \Pi^\nu / n \rfloor - 1]$
9       $\theta \leftarrow \vartheta + kn$
10      $\mathcal{P}_1 \leftarrow \mathcal{P}_1 \cup \{(h_n, \theta)\}$
11      $\mathcal{T} \leftarrow \mathcal{T} \cup \{(n, h_n, t)\}$
12  $\mathcal{S}_1' \leftarrow \mathsf{iEncode}(\mathcal{P}_1, \Pi, \nu)$
13
14  $\mathsf{sid} \leftarrow H(m \,\|\, m')$
15  $\mathcal{P}_2 \leftarrow \emptyset,\ \mathcal{R} \leftarrow \emptyset$
16  FOR ALL $(n, h_n, t) \in \mathcal{T}$:
17      IF $\mathsf{id} \notin \mathsf{prl}$:
18          $\theta \leftarrow \mathsf{iDecode}(\mathcal{S}_1, \Pi, \nu, h_n)$
19          $r \leftarrow (\theta^3 H_n(\mathsf{id}))^{2t} \bmod n$
20          $c_0 \leftarrow H(\mathsf{sid} \,\|\, r \,\|\, 0)$
21          $c_1 \leftarrow H(\mathsf{sid} \,\|\, r \,\|\, 1)$
22          $c_2 \leftarrow H(\mathsf{sid} \,\|\, r \,\|\, 2)$
23          $\mathcal{R} \leftarrow \mathcal{R} \cup \{(h_n, c_0, c_1)\}$
24      ELSE: $c_2 \leftarrow_R \{0, 1\}^\kappa$
25      $\mathcal{P}_2 \leftarrow \mathcal{P}_2 \cup \{(h_n, c_2)\}$
26  $\mathcal{S}_2' \leftarrow \mathsf{iEncode}(\mathcal{P}_2, \Pi, 1)$
27
28  $K \leftarrow 0^\kappa$, $\mathsf{groups} \leftarrow \emptyset$
29  FOR ALL $(h_n, c_0, c_1) \in \mathcal{R}$:
30      IF $\mathsf{iDecode}(\mathcal{S}_2, \Pi, 1, h_n) = c_1$:
31          $\mathsf{groups} \leftarrow \mathsf{groups} \cup \{h_n\}$
32          $K \leftarrow K \oplus c_0$
33
34  IF $\mathsf{groups} \neq \emptyset$ THEN
35      $(\mathsf{key}, \mathsf{partner}) \leftarrow (K, \mathsf{id})$
36      TERMINATE WITH "ACCEPT"
37  ELSE
38      $(\mathsf{key}, \mathsf{partner}) \leftarrow (\bot, \bot)$
39      TERMINATE WITH "REJECT"
```

Fig. 3. Specification of optimized Handshake protocol.

Observe that the exponentiation in line 7 has an a-priori known basis, namely $g = 2$. In general, diverse fast algorithms for fixed-basis exponentiations are known [24, Section 14.6]. However, we stress that in our case exponentiations' performance can be even further improved by exploiting the structure of 'square and multiply' algorithms [24], where an accumulator is repeatedly multiplied by the base element. When $g = 2$, this multiplication becomes a doubling of the accumulator, which can be implemented by a simple left-shift. The overall performance then depends solely on the cost of the squaring operation. We remark that term $(-1)^b$ for $b \in \{0, 1\}$ in line 7 should, of course, never be computed by calling an exponentiation subroutine, but by doing a distinction of cases. We conclude this paragraph by noting that messages θ (or even whole IHME structures $\mathcal{S}_1$) can be precomputed before the protocol session starts in order to further reduce run-time computations.

Note that, even though the protocol in Figure 3 is displayed as four-message protocol, by concatenating messages m' and $\mathcal{S}'_2$ into a single message, the protocol is trivially turned into a three-message protocol. We intentionally did not thoroughly specify the check of partner's pseudonym against pseudonym revocation list prl (line 17). Although in [21] revocation lists were considered of group-wide validity and managed by the corresponding authority, we can imagine other types of revocation management as well, for instance local 'per-user' revocation lists, or prls managed by third parties. The most useful type of revocation depends on the actual application. Thus, we leave the choice to the implementor.

5 Performance Analysis and Discussion

In this section we discuss the choice of practical parameters and further building blocks for a concrete implementation of our optimized Handshake protocol from Figure 3 and analyze its performance.

We start by proposing reasonable combinations of symmetric and asymmetric key sizes (κ and κ', respectively). The following table reproduces key length recommendations by NIST [27] and lists corresponding choices for Π and ν:

κ	κ'	Π	ν
80	1024	$2^\kappa + 13$	14
112	2048	$2^\kappa + 25$	20
128	3072	$2^\kappa + 51$	25
192	7680	$2^\kappa + 133$	41
256	15360	$2^\kappa + 297$	61

The hash function H (cf. Section 4.3) can be instantiated with any cryptographic hash function of output length κ. For example, SHA256 truncated to κ bits would serve in all the above cases.

As for the second hash function $H_n : \{0, 1\}^* \to \mathbb{Z}_n$, we propose the deployment of a PKCS#1.5-type padding [19], i.e. $H_n(\mathsf{id}) = s \parallel H(\mathsf{id})$, where s is some constant prefix interpreted as an integer $\bmod\, n$. Observe that in our multi-group setting we have $H_n(\mathsf{id}) = H_{n'}(\mathsf{id})$ for all ids and n, n' of the same bit length, i.e. $|n|_2 = |n'|_2$.

Finally, in Figure 4, we present performance measurements and investigate the scalability of our implemented Handshake protocol for the security level $(\kappa, \kappa') = (80, 1024)$ and varying numbers of user credentials $m = |\mathcal{C}|$ (i.e. number of different group memberships per user).

We observe that the percentage of the running time spent in IHME computations increases with growing m. This is due to the fact that IHME operations are relatively cheap but there are $O(m^2)$ of them, while exponentiations are expensive but their number is $O(m)$. Hence, only for large m the IHME workload becomes noticeable. Note that in typical applications of multi-group AH protocols, e.g. in social networks, the average number of affiliations (social groups) per user can be bounded by 100. It turns out that, in this case, the quadratic nature of IHME is still acceptable for practice.

m	5	10	25	50	100	250
total (ms)	14	29	80	188	492	2096
expos (ms)	13 (91%)	26 (90%)	65 (81%)	131 (69%)	263 (53%)	657 (31%)
IHME (ms)	1.2 (8%)	2.8 (9%)	14 (18%)	57 (30%)	229 (46%)	1438 (68%)

m	5	10	25	50	100	250
total (ms)	13	27	73	164	394	1480
expos (ms)	13 (97%)	26 (95%)	65 (89%)	131 (80%)	263 (66%)	657 (44%)
IHME (ms)	0.2 (2%)	1.2 (4%)	8 (10%)	32 (19%)	131 (33%)	823 (55%)

Fig. 4. Timing measurements of an implementation of our protocol (Figure 3), for security level $(\kappa, \kappa') = (80, 1024)$. The particular rows indicate the total time consumption of one protocol execution with m input groups, and the fraction of running time spent in exponentiation algorithms and IHME encoding/decoding, respectively. In the upper table, ν-fold IHME-encoding was used basing on Algorithm 2, while the possibility to perform IHME precomputations (Algorithm 3) was taken into account in the lower table. All timings were measured on a single core of an 'Intel XEON 2.66GHz' machine, using the gcrypt library [20] for bigint arithmetic.

6 Conclusion

We showed that affiliation-hiding authentication in the *multi-group* setting is truly practical for modern applications, where, on average, each user is a member of roughly 100 groups. We improved upon the scheme and building blocks from [21] by proposing numerous algorithmic optimizations. In particular, we explored existing solutions and developed improved algorithms for polynomial interpolation in finite fields — a major prerequisite for efficient IHME implementations. Moreover, we heavily modified the group management and handshake algorithms proposed in [21] to achieve considerably better performance in both computational means and bandwidth consumption. Our work yields a highly efficient affiliation-hiding multi-group authentication and key establishment protocol, for which implementations and appropriate performance measurements were provided.

References

1. G. Ateniese, J. Kirsch, and M. Blanton. Secret Handshakes with Dynamic and Fuzzy Matching. In *Network and Distributed System Security Symposium (NDSS 2007)*. The Internet Society, 2007.
2. D. Balfanz, G. Durfee, N. Shankar, D. K. Smetters, J. Staddon, and H.-C. Wong. Secret Handshakes from Pairing-Based Key Agreements. In *IEEE Symposium on Security and Privacy 2003*, pages 180–196. IEEE CS, 2003.
3. M. Bellare, D. Micciancio, and B. Warinschi. Foundations of Group Signatures: Formal Definitions, Simplified Requirements, and a Construction Based on General Assumptions. In *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 614–629. Springer, 2003.
4. A. Björck and V. Pereyra. Solution of Vandermonde Systems of Equations. *Mathematics of Computation*, 24(112):893–903, 1970.
5. R. Bradshaw, J. Holt, and K. Seamons. Concealing Complex Policies with Hidden Credentials. In *CCS 2004*, pages 146–157. ACM, 2004.
6. C. Castelluccia, S. Jarecki, and G. Tsudik. Secret Handshakes from CA-Oblivious Encryption. In *ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 293–307. Springer, 2004.
7. D. Chaum and E. van Heyst. Group Signatures. In *EUROCRYPT 1991*, volume 547 of *LNCS*, pages 257–265. Springer, 1991.
8. H. Cohen. *A Course in Computational Algebraic Number Theory*. Springer-Verlag New York, 1993.

9. M. Dejnakarintra and D. Banjerdpongchai. An Algorithm for Computing the Analytical Inverse of the Vandermonde Matrix. In *3rd Asian Control Conference (ASCC)*, 2000.
10. Facebook's Statistics. `http://www.facebook.com/press/info.php?statistics`.
11. R. Gennaro, H. Krawczyk, and T. Rabin. Okamoto-Tanaka Revisited: Fully Authenticated Diffie-Hellman with Minimal Overhead. In *ACNS 2010*, volume 6123 of *LNCS*, pages 309–328. Springer, 2010.
12. O. Goldreich and V. Rosen. On the Security of Modular Exponentiation with Application to the Construction of Pseudorandom Generators. *J. Cryptology*, 16(2):71–93, 2003.
13. D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography.* Springer, 2004.
14. S. Jarecki, J. Kim, and G. Tsudik. Group Secret Handshakes or Affiliation-Hiding Authenticated Group Key Agreement. In *CT-RSA 2007*, volume 4377 of *LNCS*, pages 287–308. Springer, 2007.
15. S. Jarecki, J. Kim, and G. Tsudik. Beyond Secret Handshakes: Affiliation-Hiding Authenticated Key Exchange. In *CT-RSA 2008*, volume 4964 of *LNCS*, pages 352–369. Springer, 2008.
16. S. Jarecki and X. Liu. Unlinkable Secret Handshakes and Key-Private Group Key Management Schemes. In *ACNS 2007*, volume 4521 of *LNCS*, pages 270–287. Springer, 2007.
17. S. Jarecki and X. Liu. Affiliation-Hiding Envelope and Authentication Schemes with Efficient Support for Multiple Credentials. In *ICALP (2)*, volume 5126 of *LNCS*, pages 715–726. Springer, 2008.
18. S. Jarecki and X. Liu. Private Mutual Authentication and Conditional Oblivious Transfer. In *CRYPTO 2009*, volume 5677 of *LNCS*, pages 90–107. Springer, 2009.
19. J. Jonsson and B. Kaliski. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1 (RFC 3447), 2003.
20. W. Koch. GNU Privacy Guard — The `gcrypt` Library. `http://www.gnupg.org/`.
21. M. Manulis, B. Pinkas, and B. Poettering. Privacy-Preserving Group Discovery with Linear Complexity. In *ACNS 2010*, volume 6123 of *LNCS*, pages 420–437. Springer, 2010.
22. M. Manulis, B. Poettering, and G. Tsudik. Affiliation-Hiding Key Exchange with Untrusted Group Authorities. In *ACNS 2010*, volume 6123 of *LNCS*, pages 402–419. Springer, 2010.
23. M. Manulis, B. Poettering, and G. Tsudik. Taming Big Brother Ambitions: More Privacy for Secret Handshakes. In *PETS 2010*, volume 6205 of *LNCS*, pages 149–165. Springer, 2010.
24. A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 2001.
25. P. L. Montgomery. Speeding the Pollard and Elliptic Curve Methods of Factorization. *Mathematics of Computation*, 48(177):243–264, 1987.
26. D. Naccache. Double-Speed Safe Prime Generation. `http://eprint.iacr.org/2003/175.pdf`.
27. National Institute of Standards and Technology (NIST). SP 800-57 Part 1, Recommendation for Key Management, 2007.
28. M. Schatzman. *Numerical Analysis: A Mathematical Introduction.* Clarendon Press, Oxford, 2002.
29. G. Tsudik and S. Xu. A Flexible Framework for Secret Handshakes. In *PETS 2006*, volume 4258 of *LNCS*, pages 295–315. Springer, 2006.
30. D. Vergnaud. RSA-Based Secret Handshakes. In *Intl. Workshop on Coding and Cryptography (WCC 2005)*, volume 3969 of *LNCS*, pages 252–274. Springer, 2005.
31. H. J. Wertz. On the Numerical Inversion of a Recurrent Problem: The Vandermonde Matrix. In *IEEE Transactions on Automatic Control*, 1965.
32. S. Xu and M. Yung. k-Anonymous Secret Handshakes with Reusable Credentials. In *CCS 2004*, pages 158–167. ACM, 2004.

A Computing the Inverse of a Vandermonde Matrix

Explicit formulae for the computation of the inverse $V^{-1} = (m_{1,1}, \ldots, m_{n,n}) \in \mathbb{F}^{n \times n}$ of a Vandermonde matrix $V = V(x_1, \ldots, x_n)$ (cf. Section 3.4) are sketched by Wertz in [31] and worked out by Dejnakarintra and Banjerdpongchai in [9]. Algorithm 7 has running time $O(n^2)$ and requires $n \geq 2$.

Algorithm 7 Precomputation for Interpolation

```
Input: Elements $x_1, x_2, \ldots, x_n \in \mathbb{F}$ with $i \neq j \Rightarrow x_i \neq x_j$
Output: Square Matrix $(m_{1,1}, \ldots, m_{n,n}) \in \mathbb{F}^{n \times n}$
  $a_1 \leftarrow -(x_1 + x_2)$
  $a_2 \leftarrow x_1 x_2$
  for $i = 3$ to $n$ do
    $a_i \leftarrow -x_i a_{i-1}$
    for $m = i - 1$ downto 2 do
      $a_m \leftarrow a_m - x_i a_{m-1}$
    end for
    $a_1 \leftarrow a_1 - x_i$
  end for
  for $i = 1$ to $n$ do
    $p_i \leftarrow 1$
    for $j = 1$ to $n$ do
      if $j \neq i$ then
        $p_i \leftarrow p_i(x_i - x_j)$
      end if
    end for
  end for
  for $i = 1$ to $n$ do
    $b_1 \leftarrow 1$
    for $j = 1$ to $n - 1$ do
      $b_{j+1} \leftarrow a_j + x_i b_j$
    end for
    for $j = 1$ to $n$ do
      $m_{j,i} \leftarrow b_{n-j+1}/p_i$
    end for
  end for
```

B Simultaneous Reduction of Fractions

Suppose that multiple fractions (cf. Section 3.3) have to be reduced simultaneously, i.e. for given $(c_1, d_1), \ldots, (c_n, d_n) \in \mathbb{F} \times \mathbb{F}^{\times}$ the reductions $c_1 d_1^{-1}, \ldots, c_n d_n^{-1}$ have to be computed. This can, obviously, be done by computing all divisions independently of each other, with total computation cost of nD. In the following, we describe a technique due to Cohen [8, Algorithm 10.3.4] and Montgomery [25] for doing this job at cost $(4n - 3)M + I$. It is more efficient than the naïve way whenever $I > 4M$. We motivate Algorithm 8 by observing that the inverses of arbitrary $a_1, a_2 \in \mathbb{F}^{\times}$ can be calculated simultaneously by computing $u \leftarrow (a_1 a_2)^{-1}$ and $(a_1^{-1}, a_2^{-1}) \leftarrow (u \cdot a_2, u \cdot a_1)$. This idea is generalized to more than two variables in Algorithm 8.

Note that in Algorithm 8 fractions c_i/d_i are reduced *in place*, i.e. input variables c_i are overwritten by $c_i d_i^{-1}$. Elements d_i are left unmodified. Observe moreover that, after the first **for**

Algorithm 8 Simultaneous Reduction of Fractions

```
Input: Fractions $(c_1, d_1), \ldots, (c_n, d_n) \in \mathbb{F} \times \mathbb{F}^\times$
Output: Elements $c_1 d_1^{-1}, \ldots, c_n d_n^{-1} \in \mathbb{F}$
  $u \leftarrow d_1$
  for $k = 2$ to $n$ do
    $c_k \leftarrow c_k \cdot u$
    $u \leftarrow u \cdot d_k$
  end for
  $u \leftarrow u^{-1}$
  for $k = n$ downto $2$ do
    $c_k \leftarrow c_k \cdot u$
    $u \leftarrow u \cdot d_k$
  end for
  $c_1 \leftarrow c_1 \cdot u$
```

loop, we have $u = \prod_{k=1}^{n} d_i$. It follows that error condition $d_i = 0$, for any i, can be detected efficiently by asserting $u \neq 0$ right before the inversion step.