Everybody's a Target: Scalability in Public-Key Encryption

Benedikt Auerbach, Federico Giacon, and Eike Kiltz

Ruhr University Bochum
{first.last}@rub.de

Abstract. For $1 \leq m \leq n$, we consider a natural m-out-of-n multi-instance scenario for a public-key encryption (PKE) scheme. An adversary, given n independent instances of PKE, wins if he breaks at least m out of the n instances. In this work, we are interested in the *scaling factor* of PKE schemes, SF, which measures how well the difficulty of breaking m out of the n instances scales in m. That is, a scaling factor $\mathrm{SF} = \ell$ indicates that breaking m out of n instances is at least ℓ times more difficult than breaking one single instance. A PKE scheme with small scaling factor hence provides an ideal target for mass surveillance. In fact, the Logjam attack (CCS 2015) implicitly exploited, among other things, an almost constant scaling factor of ElGamal over finite fields (with shared group parameters).

For Hashed ElGamal over elliptic curves, we use the generic group model to argue that the scaling factor depends on the scheme's granularity. In low granularity, meaning each public key contains its independent group parameter, the scheme has optimal scaling factor $\mathrm{SF} = m$; In medium and high granularity, meaning all public keys share the same group parameter, the scheme still has a reasonable scaling factor $\mathrm{SF} = \sqrt{m}$. Our findings underline that instantiating ElGamal over elliptic curves should be preferred to finite fields in a multi-instance scenario.

As our main technical contribution, we derive new generic-group lower bounds of $\Omega(\sqrt{mp})$ on the difficulty of solving both the m-out-of-n Gap Discrete Logarithm and the m-out-of-n Gap Computational Diffie-Hellman problem over groups of prime order p, extending a recent result by Yun (EUROCRYPT 2015). We establish the lower bound by studying the hardness of a related computational problem which we call the search-by-hypersurface problem.

Table of Contents

1 Introduction

For integers $1 \leq m \leq n$, consider the following natural m-out-of-n multi-instance attack scenario for a public-key encryption scheme PKE[1]. An attacker is given n independent instances (public keys) of PKE and would like to *simultaneously break semantic security at least m out of n instances*. Note that this is a different setting from the standard, well studied, multi-user attack scenario by Bellare et al. [5]. In the (security-wise) best possible scenario, compared to a (standard) single-instance attack, running an m-out-of-n multi-instance attack is m times as difficult. However, there is no guarantee that breaking m-out-of-n instances is more difficult than breaking a single instance.

This motivates the following question:

How well does the difficulty of breaking m out of n instances of PKE scale with m?

In order to quantify scalability, we define the scaling factor (relative to a fixed security notion) of PKE as

$$\mathrm{SF}^{m,n}_{\mathsf{PKE}} = \frac{\text{resources necessary to break } m \text{ out of } n \text{ instances}}{\text{resources necessary to break 1 instance}}\,, \tag{1}$$

where "resources" refers to the running time to break PKE in the fixed security notion. Clearly, the larger $\mathrm{SF}_{\mathsf{PKE}}$, the better are the security guarantees in the multi-instance setting. The best we can hope for is $\mathrm{SF}^{m,n}_{\mathsf{PKE}} = m$, meaning that breaking m out of n instances amounts to breaking m times a single instance of PKE.

Scaling Factor and Mass Surveillance. In 2012, James Bamford wrote in Wired:

> According to another top official also involved with the program, the NSA made an enormous breakthrough several years ago in its ability to cryptanalyze, or break, unfathomably complex encryption systems employed by not only governments around the world but also many average computer users in the US. The upshot, according to this official: **"Everybody's a target; everybody with communication is a target."**

This statement should appear as a surprise to the cryptographic community: Parameters for cryptographic schemes are usually chosen to make even compromising a single user a daunting challenge, meaning multi-instance attacks seem out of scope even for adversaries with nation-state capabilities. Unfortunately, the use of outdated parameters is a widespread occurrence in practice [2,16], either as a consequence of legacy infrastructure or hardware restrictions. In this case, a bad scaling factor would tip the scale from single compromised users to full-scale mass surveillance. Even more so, the hardness of several common number-theoretic problems is known to scale sub-optimally in the number of instances. Examples are factoring [9] and computing discrete logarithms in the finite-field [3,4] and elliptic-curve [14,17,18] setting. This sub-optimal scaling is typically inherited by the corresponding cryptographic schemes. It has been exploited in practice by the famous Logjam attack [2], where the authors break many Diffie-Hellman instances in TLS with nearly the same resources as to break a single Diffie-Hellman instance. Concretely, the Logjam attack could successfully break multiple 512-bit finite-field instances, and the authors also speculate about the feasibility of breaking 1024-bit instances. With our work we aim to deliver positive results by computing (non-trivial lower bounds on) the scaling factors of concrete encryption schemes that are currently employed in practice, thereby providing bounds on the hardness of performing mass surveillance.

Considered Encryption Schemes. We are able to provide non-trivial bounds on the scaling factor for Hashed ElGamal (HEG), aka. DHIES [1], in the elliptic curve ($\mathsf{HEG}[\mathsf{GGen}_{\mathbb{E}(\mathbb{F}_\ell)}]$) and the finite field ($\mathsf{HEG}[\mathsf{GGen}_{\mathbb{F}^*_\ell}]$) setting, the arguably most widely used discrete-logarithm type encryption schemes. Here $\mathsf{GGen}_{\mathbb{E}(\mathbb{F}_\ell)}$ and $\mathsf{GGen}_{\mathbb{F}^*_\ell}$ are group-generating algorithms that generate prime-order subgroups of elliptic curves and finite fields respectively. In both cases, ℓ denotes randomly chosen primes of appropriate size. We consider both schemes instantiated in low, medium, and high granularity, leading to 6 schemes $\mathsf{HEG}[\mathsf{GGen}_{\mathbb{E}(\mathbb{F}_\ell)}, \mathtt{low}]$, $\mathsf{HEG}[\mathsf{GGen}_{\mathbb{E}(\mathbb{F}_\ell)}, \mathtt{med}]$, $\mathsf{HEG}[\mathsf{GGen}_{\mathbb{E}(\mathbb{F}_\ell)}, \mathtt{high}]$, $\mathsf{HEG}[\mathsf{GGen}_{\mathbb{F}^*_\ell}, \mathtt{low}]$, $\mathsf{HEG}[\mathsf{GGen}_{\mathbb{F}^*_\ell}, \mathtt{med}]$, and $\mathsf{HEG}[\mathsf{GGen}_{\mathbb{F}^*_\ell}, \mathtt{high}]$ offering different trade-offs between public key sizes and scalability. The term *granularity* specifies which parts of the scheme's parameters belong to the global system parameters (shared among all n users), and which parts belong to the individual, user-specific public keys. Table 1 depicts the shared public system parameters and individual keys in a multi-instance setting with n parties for HEG at different granularities.

[1] Formally, in this work we consider key-encapsulation mechanisms.

PKE	Setting	Shared params	Public key pk_i
$\mathsf{HEG}[\mathsf{GGen}_{\mathbb{E}(\mathbb{F}_\ell)}, \texttt{high}]$	Elliptic curve	$\mathbb{E}(\mathbb{F}_\ell), p, g$	g^{x_i}
$\mathsf{HEG}[\mathsf{GGen}_{\mathbb{E}(\mathbb{F}_\ell)}, \texttt{med}]$	Elliptic curve	$\mathbb{E}(\mathbb{F}_\ell), p$	$g_i, g_i^{x_i}$
$\mathsf{HEG}[\mathsf{GGen}_{\mathbb{E}(\mathbb{F}_\ell)}, \texttt{low}]$	Elliptic curve	–	$\mathbb{E}_i(\mathbb{F}_{\ell_i}), p_i, g_i, g_i^{x_i}$
$\mathsf{HEG}[\mathsf{GGen}_{\mathbb{F}_\ell^*}, \texttt{high}]$	Finite field	$\mathbb{F}_\ell^*, p, g$	g^{x_i}
$\mathsf{HEG}[\mathsf{GGen}_{\mathbb{F}_\ell^*}, \texttt{med}]$	Finite field	$\mathbb{F}_\ell^*, p$	$g_i, g_i^{x_i}$
$\mathsf{HEG}[\mathsf{GGen}_{\mathbb{F}_\ell^*}, \texttt{low}]$	Finite field	–	$\mathbb{F}_{\ell_i}, p_i, g_i, g_i^{x_i}$

Table 1. Shared public system parameters and individual public keys for schemes $\mathsf{HEG}[\mathsf{GGen}_{\mathbb{E}(\mathbb{F}_\ell)}, \texttt{gran}]$ and $\mathsf{HEG}[\mathsf{GGen}_{\mathbb{F}_\ell^*}, \texttt{gran}]$ at different granularities. Here g generates a subgroup of prime order p of either an elliptic curve $\mathbb{E}(\mathbb{F}_\ell)$ or a finite field $\mathbb{F}_\ell^*$ and ℓ is a prime.

1.1 Our Results

Formal Definitions: Multi-Instance Security. The notion of n-out-of-n multi-instance security for any $n \geq 1$ was first considered and formally defined by Bellare et al. [6] in the setting of secret-key encryption. As our first contribution, we extend their notion to m-out-of-n multi-instance security for public-key encryption, for arbitrary $1 \leq m \leq n$. In fact, we give two different notions, modeling (m, n)-CPA (passive) and (m, n)-CCA (active) security.

Our (m, n)-CPA experiment provides the adversary with n independent public keys $pk[1], \ldots, pk[n]$. Next, it picks n independent challenge bits $b[1], \ldots, b[n]$ and grants the adversary access to oracle $\mathrm{Enc}(\cdot, \cdot, \cdot)$ which, given i, M_0, M_1, returns an encryption of message $M_{b[i]}$ under $pk[i]$. The adversary outputs a single bit b' together with a list $L \subseteq \{1, \ldots, n\}$ of cardinality at least m. The advantage function is defined as

$$\mathrm{Adv}^{(m,n)\text{-cpa}}_{\mathsf{PKE}} = \Pr\left[b' = \bigoplus_{i \in L} b[i]\right] - \frac{1}{2} \ .$$

That is, the adversary wins if it guesses correctly the XOR of at least m (out of n) challenge bits. (Note that the standard multi-user security notion [5] is different: Most importantly, in [5] there exists only a single challenge bit, in particular limiting this notion to the case of $m = 1$.) Why using XOR for defining the winning condition? Bellare et al. [6] argue that this is a natural metric because its well-known "sensitivity" means that even if the adversary figures out $m - 1$ of the challenge bits, it will have low advantage unless it also figures out the last. They further argue that other possible winning conditions such as using AND[2] are less natural and lead to inconsistencies. We refer to [6] for an extensive discussion. In (m, n)-CCA security, the adversary is furthermore provided with a decryption oracle $\mathrm{Dec}(\cdot, \cdot)$ which given i, c returns a decryption of c under $sk[i]$. To expand on the characteristics of the multi-instance setting, we determine the relations between the security notions (m, n)-CPA and (m, n)-CCA for different values of n, m.

Scaling Factor of $\mathsf{HEG}[\mathsf{GGen}_{\mathbb{E}(\mathbb{F}_\ell)}, \cdot]$ and $\mathsf{HEG}[\mathsf{GGen}_{\mathbb{F}_\ell^*}, \cdot]$. In order to give a lower bound on $\mathrm{SF}^{m,n}_{\mathsf{PKE}}$ as defined in Eq. (1), we need to lower bound the numerator (i.e., resources required to break m out of n instances) for all possible adversaries and upper bound the denominator (i.e., resources needed to break one instance) by specifying a concrete adversary. Unfortunately, unless the famous P vs. NP problem is settled, all meaningful lower bounds on the resources will require either an unproven complexity assumption or a restricted model of computation. We rely on the generic group model [20] for $\mathsf{HEG}[\mathsf{GGen}_{\mathbb{E}(\mathbb{F}_\ell)}, \cdot]$ (which is considered to be meaningful for elliptic-curve groups) and on a hypothesis on the best running time of variants of the number field sieve for $\mathsf{HEG}[\mathsf{GGen}_{\mathbb{F}_\ell^*}, \cdot]$ based on the fastest known attacks on finite fields.

Our main results regarding the scaling factor $\mathrm{SF}^{m,n}_{\mathsf{HEG}}$ in different granularities relative to (m, n)-CCA security are summarized in Table 2. In both considered group instantiations, HEG shows the same asymptotic scaling behavior for high and medium granularity. Thus, discrepancies in the scaling of high-granularity HEG can not be solved by switching to the less efficient medium-granularity version. In both cases however, HEG scales better in the low-granularity case. Concretely, Hashed ElGamal over

[2] I.e., by letting the adversary output a vector $b'[1], \ldots, b'[n]$ and a set I and defining the advantage function as $\mathrm{Adv}^{(m,n)\text{-cpa}}_{\mathsf{PKE}} = \Pr[\bigwedge_{i \in I} b[i] = b'[i]] - 1/2^m$.

PKE	Setting	Scaling factor
$\mathsf{HEG}[\mathsf{GGen}_{\mathbb{E}(\mathbb{F}_\ell)}, \{\mathtt{high}, \mathtt{med}\}]$	Elliptic curve	$\Theta(\sqrt{m})$
$\mathsf{HEG}[\mathsf{GGen}_{\mathbb{E}(\mathbb{F}_\ell)}, \mathtt{low}]$	Elliptic curve	$\Theta(m)$
$\mathsf{HEG}[\mathsf{GGen}_{\mathbb{F}_\ell^*}, \{\mathtt{high}, \mathtt{med}\}]$	Finite field	$\begin{cases} 1 & \delta \leq 0.67 \\ L_\ell(1/3, \delta - 0.67) & \delta > 0.67 \end{cases}$
$\mathsf{HEG}[\mathsf{GGen}_{\mathbb{F}_\ell^*}, \mathtt{low}]$	Finite field	$\begin{cases} L_\ell(1/3, \delta) & 0 \leq \delta < 0.105 \\ L_\ell(1/3, 0.105) & 0.105 \leq \delta < 0.368 \\ L_\ell(1/3, -0.263 + \delta) & 0.368 \leq \delta \end{cases}$

Table 2. Lower bounds on the scaling factor $\mathrm{SF}^{m,n}_{\mathsf{HEG}}$ relative to (m,n)-CCA security. $L_\ell(1/3, c)$ is defined as $\exp((c + o(1))(\log \ell)^{1/3}(\log\log \ell)^{2/3})$. In the finite field case $m = L_\ell(1/3, \delta)$ for some $\delta \geq 0$.

elliptic curves (modeled as generic groups) scales optimally for low-granularity parameters. For medium and high granularity, on the other hand, the scaling factor is of order $\Theta(\sqrt{m})$, where the constants hidden by the Θ-notation are small.

Let $L_\ell(1/3, c) \coloneqq \exp((c + o(1))(\log \ell)^{1/3}(\log\log \ell)^{2/3})$. For HEG in the finite field setting with respect to high and medium granularity, we see that the scaling factor is roughly 1 for up to $m = L_\ell(1/3, 0.67)$ instances. Beyond, the KEM scales linearly with slope $L_\ell(1/3, -0.67)$. Note that $L_\ell(1/3, 0.67)$ is large for typical values of ℓ. Concretely, for 512 bit primes we get that $L_\ell(1/3, 0.67) \approx 2^{22}$ meaning that the effort of breaking 2^{22} instances roughly equals the effort to break a single instance. While the concrete number is obtained ignoring the $o(1)$ terms in L_ℓ, it still matches the empirical results of [2, Table 2]. For low granularity and for up to $L_\ell(1/3, 0.108)$ instances, $\mathsf{HEG}[\mathsf{GGen}_{\mathbb{F}_\ell^*}, \mathtt{low}]$ scales optimally. For $L_\ell(1/3, 0.108) \leq m \leq L_\ell(1/3, 0.368)$, the scaling factor is roughly constant, and for larger numbers of instances, it scales linearly with slope $L_\ell(1/3, -0.263)$.

Derivation of the Scaling Factors. As we will explain below in more detail, the bounds from Table 2 are obtained in two steps. In a **first step**, we consider an m-out-of-n multi-instance version of the the Gap Computational Diffie-Hellman problem, (m,n)-GapCDH$[\mathsf{GGen}, \mathtt{gran}]$, where the term "gap" refers to the presence of a Decisional Diffie-Hellman (DDH) oracle. The following theorem holds for all $\mathsf{GGen} \in \{\mathsf{GGen}_{\mathbb{E}(\mathbb{F}_\ell)}, \mathsf{GGen}_{\mathbb{F}_\ell^*}\}$ and $\mathtt{gran} \in \{\mathtt{high}, \mathtt{med}, \mathtt{low}\}$.

Theorem *If* (m,n)-GapCDH$[\mathsf{GGen}, \mathtt{gran}]$ *is hard, then* $\mathsf{HEG}[\mathsf{GGen}, \mathtt{gran}]$ *is* (m,n)-CCA *secure, tightly.*

The theorem (described formally in Section 4) is a somewhat straightforward generalization of [1]. We stress that tightness in our previous theorem is an essential ingredient to obtain overall tight bounds on the scaling factor.

In a **second step**, we provide bounds on the (m,n)-GapCDH$[\mathsf{GGen}, \mathtt{gran}]$ problem. In the finite field case, we rely on the following hypothesis:

Hypothesis 1 *The fastest algorithms to break* (m,n)-GapCDH$[\mathsf{GGen}_{\mathbb{F}_\ell^*}, \mathtt{gran}]$ *are variants of the number field sieve [3,4] which require running time*

$$T = \begin{cases} L_\ell(1/3, 1.902) + m \cdot L_\ell(1/3, 1.232) & \mathtt{gran} \in \{\mathtt{high}, \mathtt{med}\} \\ \min\{m \cdot L_\ell(1/3, 1.902), L_\ell(1/3, 2.007) + m \cdot L_\ell(1/3, 1.639)\} & \mathtt{gran} = \mathtt{low} \end{cases}.$$

The lower bounds on $\mathrm{SF}^{m,n}$ for $\mathsf{HEG}[\mathsf{GGen}_{\mathbb{F}_\ell^*}, \mathtt{gran}]$ are obtained by combining the previous theorem and Hypothesis 1. The running times specified in the hypothesis stem from the multi-field NFS [4] (high/medium granularity) and the DLOG factory [3] (low granularity). Both variants first require an instance-independent precomputation. Then instances can be solved with a constant computational effort. The values $\delta = 0.67$ and $\delta = 0.368$ of Table 2 correspond to the number of instances starting from which the cumulative cost of breaking the instances outweighs the cost of the precomputation.

In the elliptic-curve case, we make the hypothesis that the fastest adversary attacking the system is a generic-group adversary. Concretely, we provide lower bounds on (m,n)-GapCDH$[\mathsf{GGen}_{\mathsf{gg}}, \mathtt{gran}]$ in different granularities, where $\mathsf{GGen}_{\mathsf{gg}}$ generates a generic group [20] of prime-order p. We do so by first relating it to a m-out-of-n multi-instance version of the Gap Discrete Logarithm problem,

m-out-of-n problem	Given	Break m out of	Gap?	Complexity	Ref.
DL[GGen, high]	$\mathbb{G}, p, g, g^{x_1}, \ldots, g^{x_n}$	$x_1, \ldots, x_n$	–	$\Theta(\sqrt{mp})$	[23,22,18]
DL[GGen, med]	$\mathbb{G}, p, g_1, g_1^{x_1}, \ldots, g_n, g_n^{x_n}$	$x_1, \ldots, x_n$	–	$\Theta(\sqrt{mp})$	§5.3
DL[GGen, low]	$\mathbb{G}_1, p_1, g_1, g_1^{x_1}, \ldots, \mathbb{G}_n, p_n, g_n, g_n^{x_n}$	$x_1, \ldots, x_n$	–	$\Theta(m\sqrt{p})$	§5.4
GapDL[GGen, high]	$\mathbb{G}, p, g, g^{x_1}, \ldots, g^{x_n}$	$x_1, \ldots, x_n$	✓	$\Theta(\sqrt{mp})$	§5.2
GapDL[GGen, med]	$\mathbb{G}, p, g_1, g_1^{x_1}, \ldots, g_n, g_n^{x_n}$	$x_1, \ldots, x_n$	✓	$\Theta(\sqrt{mp})$	§5.3
GapDL[GGen, low]	$\mathbb{G}_1, p_1, g_1, g_1^{x_1}, \ldots, \mathbb{G}_n, p_n, g_n, g_n^{x_n}$	$x_1, \ldots, x_n$	✓	$\Theta(m\sqrt{p})$	§5.4
GapCDH[GGen, high]	$\mathbb{G}, p, g, g^{x_1}, g^{y_1}, \ldots, g^{x_n}, g^{y_n}$	$g^{x_1 y_1}, \ldots, g^{x_n y_n}$	✓	$\Theta(\sqrt{mp})$	§6.1
GapCDH[GGen, med]	$\mathbb{G}, p, g_1, g_1^{x_1}, g_1^{y_1}, \ldots, g_n, g_n^{x_n}, g_n^{y_n}$	$g_1^{x_1 y_1}, \ldots, g_n^{x_n y_n}$	✓	$\Theta(\sqrt{mp})$	§6.2
GapCDH[GGen, low]	$\mathbb{G}_1, p_1, g_1, g_1^{x_1}, g_1^{y_1}, \ldots, \mathbb{G}_n, p_n, g_n, g_n^{x_n}, g_n^{y_n}$	$g_1^{x_1 y_1}, \ldots, g_n^{x_n y_n}$	✓	$\Theta(m\sqrt{p})$	§6.3

Table 3. Definition and generic-group complexity of problems (m,n)-DL[GGen, gran], (m,n)-GapDL[GGen, gran], and (m,n)-GapCDH[GGen, gran], where gran belongs to {low, med, high}. $\mathbb{G}$ and $\mathbb{G}_i$ are generic groups of prime order p and p_i, with generators g and g_i, respectively. The third column defines the problem's winning condition. The Gap column indicates the presence of a DDH oracle.

(m,n)-GapDL[$\mathsf{GGen}_{\mathsf{gg}}$, gran]. We furthermore extend Yun's known lower bounds in the generic group model for (m,n)-DL[$\mathsf{GGen}_{\mathsf{gg}}$, high] [23,22,18] to the gap setting and to (m,n)-GapDL[$\mathsf{GGen}_{\mathsf{gg}}$, gran] for different granularities gran $\in$ {high, med, low}. Concretely, we will prove the following main theorem.

Theorem *The best generic algorithm to break* (m,n)-GapCDH[$\mathsf{GGen}_{\mathsf{gg}}$, gran] *requires running time*

$$T = \begin{cases} \Theta(\sqrt{mp}) & \texttt{gran} \in \{\texttt{high}, \texttt{med}\} \\ \Theta(m\sqrt{p}) & \texttt{gran} = \texttt{low} \end{cases},$$

and the constant hidden by the Θ *notation are small (between* 0.1 *and* 6.6*).*

The lower bounds on $\mathrm{SF}^{m,n}$ for HEG[$\mathsf{GGen}_{\mathbb{E}(\mathbb{F}_\ell)}$, gran] are obtained by combining our previous theorems and assuming that elliptic curve groups behave like generic groups.

1.2 Generic Bounds on Multi-Instance GapCDH: Technical Details

As a technical tool to prove the security of Hashed ElGamal, we consider the multi-instance variants of three different problems: discrete logarithm ((m,n)-DL[$\mathsf{GGen}_{\mathsf{gg}}$, gran]), gap discrete logarithm ((m,n)-GapDL[$\mathsf{GGen}_{\mathsf{gg}}$, gran]), and gap computational Diffie-Hellman ((m,n)-GapCDH[$\mathsf{GGen}_{\mathsf{gg}}$, gran]) in different granularities, see Table 3.

We now discuss the complexity column of Table 3. It is well known that the running time of solving (m,n)-DL[$\mathsf{GGen}_{\mathsf{gg}}$, high] is $\Theta(\sqrt{mp})$, the lower bound being in the generic group model [23,22], the matching upper bound stemming from a concrete generic algorithm [18]. It is not hard to see that the bounds on (m,n)-DL[$\mathsf{GGen}_{\mathsf{gg}}$, med] are basically the same because the generators g_i can be viewed as "high-granularity instances" g^{x_j}. As for (m,n)-DL[$\mathsf{GGen}_{\mathsf{gg}}$, low], the adversary essentially has to solve m discrete logarithms in n independent generic groups $\mathbb{G}_1, \ldots, \mathbb{G}_n$, which results in a lower bound of $\Omega(m\sqrt{p})$.

Our **first main technical result** (Corollary 1) is a non-trivial extension of Yun's generic lower bound [23,22] to the gap setting, i.e., a new lower bound of $\Omega(\sqrt{mp})$ on solving the (m,m)-GapDL[$\mathsf{GGen}_{\mathsf{gg}}$, high].

Our **second main technical result** (Theorem 5) states that, in high granularity, the (m,m)-GapDL and the (m,n)-GapCDH problems are essentially equally hard in the algebraic group model [12], hence implying the required bounds in the generic group model. The results in medium and low granularity follow as in the discrete logarithm setting.

Main Technical Result 1: Lower Bound on (m,m)-GapDL[$\mathsf{GGen}_{\mathsf{gg}}$, high]. We define a new "hard" problem called the *polycheck discrete logarithm problem*: The security game is the same as that of standard multi-instance DL, but the adversary has additional access to an oracle Eval that behaves as follows. Given as input to Eval a polynomial $f \in \mathbb{Z}_p[X_1, \ldots, X_k]$ and group elements $g^{x_1}, \ldots, g^{x_k}$, it returns 1 if and only if $g^{f(x_1,\ldots,x_k)} = 1$. This problem is easier than GapDL: In fact, we can simulate the gap

oracle $\mathrm{DDH}(g^x, g^y, g^z)$ by querying $\mathrm{Eval}(f := X_1X_2 - X_3, g^x, g^y, g^z)$. In the generic group model, we can bound the advantage of an adversary against the m-out-of-m polycheck discrete logarithm problem that queries polynomial of degree at most d ((m,m)-d-PolyDL[$\mathsf{GGen}_{\mathtt{gg}}, \mathtt{high}$]) as

$$\mathrm{Adv}^{(m,m)\text{-}d\text{-polydl}} \lesssim \left(\frac{dq^2 + dq_{\mathrm{Eval}}}{mp}\right)^m ,$$

where q bounds the queries to the group-operation oracle, q_{Eval} to Eval, and p is the order of the generic group. The bound for high-granularity GapDL follows by setting $d = 2$.

The result is proven by extending the arguments by Yun [23] for the standard multi-instance DL problem. In line with Yun's approach, we define the *search-by-hypersurface* problem in dimension m (m-$\mathrm{SHS}_d[p]$), which requires to find a uniformly sampled point $\boldsymbol{a} \in \mathbb{Z}_p^m$ while being able to check whether $\boldsymbol{a}$ is a zero of adaptively chosen polynomials in $\mathbb{Z}_p[X_1, \ldots, X_m]$ for polynomials of degree at most d. Notably, Yun's *search-by-hyperplane-queries* problem in dimension m is equivalent to m-SHS_1.

We show that any generic adversary against (m,m)-d-PolyDL[$\mathsf{GGen}_{\mathtt{gg}}, \mathtt{high}$] can be transformed into an adversary against m-SHS_d, and then proceed to bound the advantage of an adversary against m-SHS_d. The key step is observing that an adversary can make at most m useful hypersurface queries, that is, queries that return 1 (hence, identify a hypersurface on which the point $\boldsymbol{a}$ lies) and whose output is not easy to determine based on previous queries. The key difference between our result and Yun's lies in how useful queries are processed and counted. Since Yun considers only polynomials of degree 1, a hypersurface defined by a polynomial of degree 1 is a hyperplane of the affine space $\mathbb{Z}_p^m$. Each useful query identifies another hyperplane on which the sought point lies. When intersecting another hyperplane with the intersection of the hyperplanes previously found, the dimension of the intersection as an affine subspace is brought down by one. The dimension of the full affine space being m, at most m such queries can be made before identifying a single point (dimension 0). However, generalizing to hypersurfaces generated by polynomials of degree ≥ 2 requires to carry over more sophisticated arguments from algebraic geometry. Firstly, intersecting m hypersurfaces does not, in general, identify a single point. Secondly, intersection of two hypersurfaces might give rise to the union of two or more irreducible components. Intersecting further with a hypersurface containing just one of those irreducible components would qualify as a useful query, however would not bring down the dimension of the intersection by one. This impasse is overcome by guessing the correct component at each step. Fortunately, Bézout's theorem and a discerning choice of the guessing probabilities at each useful query makes the argument go through with just an additional loss of d^m, which is absorbed by the exponential bound in the dimension.

Main Technical Result 2: (m,m)-GapDL[$\mathsf{GGen}, \mathtt{high}$] Implies (m,n)-GapCDH[$\mathsf{GGen}, \mathtt{high}$]. The algebraic group model, introduced by Fuchsbauer et al. [12], essentially treats all algorithms as algebraic, in the sense that for every group element X they output they also have to provide a representation $X = \prod Y_i^{a_i}$ in terms of all previously observed group elements $Y_1, \ldots, Y_k$. Importantly, a bound in the generic group model can be extended through an algebraic reduction to a different problem.

Our second technical result (Theorem 5) presents an algebraic reduction between the problems (m,n)-GapCDH[$\mathsf{GGen}, \mathtt{high}$] and (m,m)-GapDL[$\mathsf{GGen}, \mathtt{high}$] with a tightness loss of 2^m in the algebraic group model. Combining this with the generic-group lower bound we prove as our first main technical result, we obtain, in the generic group model:

$$\mathrm{Adv}_{\mathtt{high}}^{(m,n)\text{-gcdh}} \overset{\text{Th. 5}}{\leq} 2^m \cdot \mathrm{Adv}_{\mathtt{high}}^{(m,m)\text{-gdl}} \overset{\text{Cor. 1}}{\lesssim} 2^m \left(\frac{q^2 + q_{\mathrm{DDH}}}{mp}\right)^m \approx \left(\frac{2q^2}{mp}\right)^m ,$$

where q bounds the queries to the group-operation oracle, q_{DDH} to the gap oracle, and p is the order of the generic group. Note that the reduction's exponential loss of 2^m gets swallowed by the (m,m)-GapDL[$\mathsf{GGen}_{\mathtt{gg}}, \mathtt{high}$] bound. More importantly, by the above bound one requires $q \geq \Omega(\sqrt{mp})$ generic-group operations to break (m,n)-GapCDH[$\mathsf{GGen}_{\mathtt{gg}}, \mathtt{high}$] with overwhelming advantage.

A natural approach to tackle the proof of Theorem 5 would be to adapt the single-instance proof presented in [12] to the multi-instance setting. By following this strategy, however, the reduction would need to solve multivariate systems of quadratic equations, which is in general believed to be a hard problem. The proof techniques we employ are in fact significantly different.

The path we pursue maintains, instead, the linear character of the system. The reduction distributes the i-th DL challenges in either the X or Y components of the i-th challenges to the CDH adversary. The

intuition at the core of the proof is that an adversary finding the CDH solution for any one instance must provide the DL of at least one of the two corresponding challenge components (even if possibly depending on the remaining, unrecovered DLs). If the reduction manages to embed the m DL challenges at the right spot, then it is able to recover all logarithms. The reduction loss of 2^m is consequence of this guess. Moreover, expanding the m DL challenges into n CDH challenges adds a further layer of complexity. Any wrong choice at this step would undermine either the reduction's ability to extract the discrete logarithms or give to the adversary information on where the DL challenges are embedded.

1.3 Future Directions

Corrigan-Gibbs and Kogan [11] consider the multi-instance discrete logarithm problem in a setting where the adversary is allowed to first perform unbounded preprocessing over the group to produce an advice string of bounded size, which in a second stage is used to solve multiple discrete logarithm instances. The resulting lower bounds in the generic group model were also derived by Coretti et al. [10] using a different technique. It would be interesting to compute scaling factors of the considered schemes taking preprocessing into account. Another possible direction is to derive lower bounds on the scaling factor for practical encryption schemes in the RSA setting (e.g., RSA-OAEP [7]) and in the post-quantum setting (e.g., based on lattices and codes).

2 Preliminaries

2.1 Notation

Vector Notation. We denote vectors with boldface fonts, for example $\boldsymbol{v}$. The number of elements of a vector is represented by $|\boldsymbol{v}|$. Element indexing starts from 1, and the entry at position i is accessed through square brackets: $\boldsymbol{v}[i]$. To initialize all entries of a vector to some element a we write $\boldsymbol{v}[\cdot] \leftarrow a$. We may initialize multiple vectors simultaneously, and moreover initialize them through running some (possibly randomized) routine. As an example, we could initialize a vector of public and of secret keys as $(\boldsymbol{pk}, \boldsymbol{sk})[\cdot] \leftarrow_{\$} \mathsf{Gen}$ to indicate that for every index i we run Gen with fresh randomness and, denoting the output with (pk, sk), set $\boldsymbol{pk}[i] \leftarrow pk$ and $\boldsymbol{sk}[i] \leftarrow sk$. Given any set of indices I, we denote with $\boldsymbol{v}[I]$ the vector that contains only the entries indexed with elements in I. For example, if $\boldsymbol{v} = (a, b, c)$ then $\boldsymbol{v}[\{1, 3\}] = (a, c)$. We slightly abuse this notation, writing $\boldsymbol{v}[I] \leftarrow \boldsymbol{w}$ when replacing each entry of $\boldsymbol{v}$ whose indices belong to I by the elements of $\boldsymbol{w}$ in their order. For example, if $\boldsymbol{v} = (a, b, c)$ and we execute $\boldsymbol{v}[\{1, 3\}] \leftarrow (d, e)$ then $\boldsymbol{v} = (d, b, e)$.

Group Notation. In this paper we consider groups $\mathbb{G}$ of prime order p, generated by g. We call $\mathcal{G} = (\mathbb{G}, p, g)$ a group representation. A group-generating algorithm GGen is a randomized algorithm that outputs a group representation $\mathcal{G}$. We assume that all groups output by GGen are of the same bit length.

In this work we consider two instantiations $\mathsf{GGen}_{\mathbb{E}(\mathbb{F}_\ell)}$ and $\mathsf{GGen}_{\mathbb{F}_\ell^*}$ of group-generating algorithms. In both cases ℓ denotes a randomly sampled prime of appropriate size. Group descriptions $\mathcal{G}$ output by $\mathsf{GGen}_{\mathbb{E}(\mathbb{F}_\ell)}$ are prime-order p subgroups of elliptic curves defined over the field $\mathbb{F}_\ell$. Group descriptions output by the second considered group-generating algorithm $\mathsf{GGen}_{\mathbb{F}_\ell^*}$ are subgroups of the multiplicative group $\mathbb{F}_\ell^*$ of sufficiently large prime order.

Except for the group generators, all group elements will be denoted with uppercase letters, e.g., X. We use vectors and matrices of elements in $\mathbb{Z}_p$ to compute with group elements: If Y is a group element and $\boldsymbol{x}$ is a vector of elements in $\mathbb{Z}_p$, we write $Y^{\boldsymbol{x}}$ to denote the group element vector $(Y^{\boldsymbol{x}[1]}, Y^{\boldsymbol{x}[2]}, \ldots)$. Similarly, given some matrix $M = (m_{ij})_{i,j \in [1\,..\,n] \times [1\,..\,k]}$ and a vector of group elements $\boldsymbol{Y}$ of size k, we define $\boldsymbol{Y}^M$ to be the n-size vector $(\boldsymbol{Y}[1]^{m_{11}} \ldots \boldsymbol{Y}[k]^{m_{1k}}, \ldots, \boldsymbol{Y}[1]^{m_{n1}} \ldots \boldsymbol{Y}[k]^{m_{nk}})$. Note that if $\boldsymbol{Y} = g^{\boldsymbol{y}}$ then $\boldsymbol{Y}^M = g^{M\boldsymbol{y}}$.

Security Games. We define security notions via *code-based games* [8]. A game G consists of a main procedure and zero or more oracles that can be accessed from within the game. The game is defined with respect to an adversary $\mathcal{A}$, which is invoked within the main procedure. The adversary may have access to some of the oracles of the game: The ability to access oracle O is represented by invoking the adversary as $\mathcal{A}^{\mathsf{O}}$. When the game stops, it outputs either a success (1) or a failure (0) symbol. With $\Pr[\mathrm{G}(\mathcal{A})]$ we denote the probability that adversary $\mathcal{A}$ wins, i.e., that game G, executed with respect to $\mathcal{A}$, stops with output 1.

2.2 Generic/Algebraic Group Model

Generic Group Model. Intuitively, the Generic Group Model (GGM) is an abstraction to study the behavior of adversaries that do not exploit any specific structure of the group at play, but rather treat the group in a black-box fashion. This is usually modeled by representing group elements exclusively through "opaque" handles, which hide the structure of the group. These handles are used as input to a model-bound oracle, the group-operation oracle, which is the only interface to the group available to the adversary. An algorithm with such restrictions is referred to as a *generic algorithm*. The running time of generic adversaries is normally measured in number of calls to the group-operation oracle. For further details on the GGM we refer to [20,19]. To derive bounds on the hardness of solving certain computational problems with respect to $\mathsf{GGen}_{\mathbb{E}(\mathbb{F}_\ell)}$ we model the output elliptic curves as generic groups. For clarity, in this case we denote the group-generating algorithm by $\mathsf{GGen}_{\mathsf{gg}}$.

Algebraic Group Model. For every group element Z it returns, an *algebraic algorithm* $\mathcal{A}$ must present a description of this element in terms of the elements it has previously seen. That is, if n is the order of the group and $X_1, \ldots, X_k$ are the elements that $\mathcal{A}$ received so far from the game, then $\mathcal{A}$ must return some elements $a_1, \ldots, a_k \in \mathbb{Z}_n$ such that $Z = X_1^{a_1} \ldots X_k^{a_k}$. We use the algebraic group model to analyze generic reductions:

Note that a generic reduction executed with respect to a generic adversary is itself a generic algorithm. Without loss of generality we may assume that generic adversaries are algebraic, which allows the reduction to exploit the useful algebraic representation of the input group elements. As demonstrated by Fuchsbauer et al. [12], this idea gives a handy technique for carrying over generic lower bounds through generic reductions, as seen in the following lemma.

Lemma 1 ([12, Lemma 2.2]). *Let α, Δ be constants and let $\mathcal{R}$ be a generic reduction $\mathcal{R}$ from game G_0 to G_1. Assume that for every generic adversary $\mathcal{A}$ that succeeds with probability ε and makes at most q group-operation queries, reduction $\mathcal{R}$ executed with respect to $\mathcal{A}$ makes at most $q + \Delta$ group-operation queries and succeeds with probability of at least $\alpha\varepsilon$. If there exists a function f such that $\Pr[\mathrm{G}_1(\mathcal{B})] \leq f(q)$ for every generic adversary $\mathcal{B}$ making at most q group-operation queries, then for every generic adversary $\mathcal{A}$ making at most q group-operation queries we obtain $Pr[\mathrm{G}_0(\mathcal{A})] \leq \alpha^{-1} f(q + \Delta)$.*

2.3 Key-Encapsulation Mechanisms

A *key-encapsulation mechanism* (KEM) KEM specifies the following. Parameter generation algorithm Par generates public parameters *par* to be utilized by all users. Key-generation algorithm Gen gets the parameters as input and outputs a pair (pk, sk) consisting of a public and a secret key. Encapsulation algorithm Enc on input of the parameters and a public key outputs a pair (K, c) consisting of an encapsulated key K belonging to the encapsulated key space $\mathsf{KS}(par)$ and a ciphertext c belonging to the ciphertext space $\mathsf{CS}(par)$. Deterministic decapsulation algorithm Dec receives the parameters, a secret key sk and a ciphertext c as input and returns either the symbol $\perp$ indicating failure or an encapsulated key K. For *correctness* we require that for all *par* output of Par and for every (pk, sk) output of $\mathsf{Gen}(par)$ we obtain $K \leftarrow \mathsf{Dec}(par, sk, c)$ for $(K, c) \leftarrow_{\$} \mathsf{Enc}(par, pk)$.

3 Multi-Instance Security

In this section we investigate the m-out-of-n multi-instance security of key-encapsulation mechanisms. After giving security definitions in Section 3.1, in Section 3.2 we consider the relation between security notions for varying m and n. In Section 3.3 we define the scaling factor, which measures how well the security of KEMs scales with the number of users. Finally, in Section 3.4 we give security definitions for Diffie-Hellman type problems in the multi-instance setting, which will be used in the security analysis of the Hashed-ElGamal KEM in the next section.

3.1 Key Encapsulation in the Multi-Instance Setting

Below we give security definitions for key-encapsulation mechanisms in the multi-instance setting. Our definitions are in the xor metric introduced by Bellare et al. [6] for symmetric encryption schemes.

Games $\mathrm{G}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{A})$, $\mathrm{G}^{(m,n)\text{-cca}}_{\mathsf{KEM}}(\mathcal{A})$

$\boldsymbol{C}^*[\cdot] \leftarrow \emptyset$
$\boldsymbol{b} \leftarrow_{\$} \{0,1\}^n$
$par \leftarrow_{\$} \mathsf{Par}$
for $i \in [1\,..\,n]$:
  $(\boldsymbol{pk}[i], \boldsymbol{sk}[i]) \leftarrow_{\$} \mathsf{Gen}(par)$
$(L, b') \leftarrow_{\$} \mathcal{A}^{\mathrm{Enc}}(par, \boldsymbol{pk})$ $\quad \backslash\backslash\, (m,n)\text{-CPA}$
$(L, b') \leftarrow_{\$} \mathcal{A}^{\mathrm{Enc},\mathrm{Dec}}(par, \boldsymbol{pk})$ $\quad \backslash\backslash\, (m,n)\text{-CCA}$
if $|L| < m$: return 0
if $\oplus_{i\in L} \boldsymbol{b}[i] = b'$: return 1
else: return 0

Oracle $\mathrm{Enc}(i)$

$(K_1^*, c^*) \leftarrow_{\$} \mathsf{Enc}(par, \boldsymbol{pk}[i])$
$K_0^* \leftarrow_{\$} \mathsf{KS}(par)$
$\boldsymbol{C}^*[i] \leftarrow \boldsymbol{C}^*[i] \cup \{c^*\}$
return $(K^*_{\boldsymbol{b}[i]}, c^*)$

Oracle $\mathrm{Dec}(i, c)$

if $c \in \boldsymbol{C}^*[i]$: return $\perp$
$K \leftarrow \mathsf{Dec}(par, \boldsymbol{sk}[i], c)$
return K

Fig. 1. Games $\mathrm{G}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}$ and $\mathrm{G}^{(m,n)\text{-cca}}_{\mathsf{KEM}}$ modeling m-out-of-n multi-instance indistinguishability of encapsulated keys from random. We assume that $L \subseteq [1\,..\,n]$.

We target m-out-of-n multi-instance indistinguishability of encapsulated keys from random against chosen-plaintext attacks ((m,n)-CPA) or chosen-ciphertext attacks ((m,n)-CCA).

In its most general form, the xor metric models the inability of an adversary to break m out of n instances of a decisional problem. The adversary receives as input n challenges, generated independently of each other with respect to n independent challenge bits $\boldsymbol{b}$. The adversary's task is to output a subset $L \subseteq [1\,..\,n]$ of size at least m (representing the "broken instances") together with a guess for $\bigoplus_{i\in L} \boldsymbol{b}[i]$; the intuition being that as long as at least one of the challenge bits contained in L is hidden to the adversary, so is $\bigoplus_{i\in L} \boldsymbol{b}[i]$, reducing the adversary to guessing the final output.

Formally, let KEM be a KEM and let $m, n \in \mathbb{N}$ such that $1 \le m \le n$. Consider games $\mathrm{G}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{A})$ and $\mathrm{G}^{(m,n)\text{-cca}}_{\mathsf{KEM}}(\mathcal{A})$ of Fig. 1 associated to KEM, m, n, and an adversary $\mathcal{A}$. In both games, $\boldsymbol{b}$ is a vector of n challenge bits, which corresponds to vectors $\boldsymbol{pk}, \boldsymbol{sk}$ of public and secret keys, which are set up using a single set of global parameters par. The adversary has access to a challenge oracle Enc, which on input of index $i \in [1\,..\,n]$ returns a pair consisting of an encapsulated key and a ciphertext generated with $\mathsf{Enc}(par, \boldsymbol{pk}[i])$ if the challenge bit $\boldsymbol{b}[i]$ equals 1, or, if $\boldsymbol{b}[i]$ equals 0, a ciphertext and a randomly sampled element of $\mathsf{KS}(par)$. At the end of the game, adversary $\mathcal{A}$ outputs a list of indices $L \subseteq [1\,..\,n]$ and a bit b'. $\mathcal{A}$ wins if L contains at least m elements and if $b' = \bigoplus_{i\in L} \boldsymbol{b}[i]$. In game $\mathrm{G}^{(m,n)\text{-cca}}_{\mathsf{KEM}}(\mathcal{A})$ the adversary additionally has access to a decapsulation oracle Dec, which on input of index $i \in [1\,..\,n]$ and ciphertext c returns the decapsulation of c under parameters par and secret key $\boldsymbol{sk}[i]$ (unless c was output as response to a challenge query $\mathrm{Enc}(i)$ for index i).

We define $\mathcal{A}$'s advantage in game $\mathrm{G}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}$ and $\mathrm{G}^{(m,n)\text{-cca}}_{\mathsf{KEM}}$ respectively as

$$\mathrm{Adv}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{A}) = 2\Pr[\mathrm{G}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{A})] - 1 \ ,$$
$$\mathrm{Adv}^{(m,n)\text{-cca}}_{\mathsf{KEM}}(\mathcal{A}) = 2\Pr[\mathrm{G}^{(m,n)\text{-cca}}_{\mathsf{KEM}}(\mathcal{A})] - 1 \ .$$

The definition we have just presented lends itself naturally to a comparison with the standard multi-user security notion of Bellare et al. [5]. We describe the relationship between multi-user security and $(1,n)$-CCA in detail in Appendix A.

3.2 Advantage Relations for Different m and n

The relations between (m',n')-CPA and (m,n)-CPA security are summarized in Fig. 2. They are stated more formally in the following theorem, whose proof is in Appendix B.

Theorem 1. *Let m, n, m', n' be positive integers such that $m \le n$, $m' \le n'$, and let KEM be any KEM scheme. Then for every adversary $\mathcal{A}$ against game $\mathrm{G}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}$ there exists an adversary $\mathcal{B}$ against game $\mathrm{G}^{(m',n')\text{-cpa}}_{\mathsf{KEM}}$ such that:*

1. *If $m' \le m$ and $m'n \le mn'$ then $\mathcal{B}$ has roughly the same running time of $\mathcal{A}$ and*

$$\mathrm{Adv}^{(m',n')\text{-cpa}}_{\mathsf{KEM}}(\mathcal{B}) \ge \frac{1}{2}\mathrm{Adv}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{A}) \ .$$

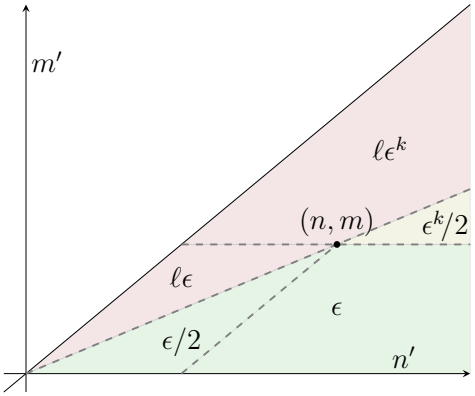

Fig. 2. Relations between (m', n')-CPA and (m, n)-CPA security. Given $\mathcal{A}$ against (m, n)-CPA with advantage ϵ, one can build $\mathcal{B}$ against (m', n')-CPA with advantage as shown in figure, depending on its position on the plane. The constants in the figure are $k = \lceil m'/m \rceil$ and $\ell = \frac{1}{2}\binom{n'}{m'}\binom{\lceil nm'/m \rceil}{m'}^{-1}$. The same result holds for CCA.

Additionally, if $n' - m' \geq n - m$ then the reduction does not lose the factor $1/2$.

2. *If $m' \leq m$ and $m'n > mn'$ then $\mathcal{B}$ has roughly the same running time of $\mathcal{A}$ and*

$$\mathrm{Adv}^{(m',n')\text{-cpa}}_{\mathsf{KEM}}(\mathcal{B}) \geq \frac{1}{2}\binom{n'}{m'}\binom{\lceil nm'/m \rceil}{m'}^{-1} \mathrm{Adv}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{A}) \ .$$

3. *If $m' > m$ and $m'n \leq mn'$ then $\mathcal{B}$ has roughly $k = \lceil m'/m \rceil$ times the running time of $\mathcal{A}$ and*

$$\mathrm{Adv}^{(m',n')\text{-cpa}}_{\mathsf{KEM}}(\mathcal{B}) \geq \frac{1}{2}\left(\mathrm{Adv}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{A})\right)^k \ .$$

Additionally, if m divides m' then the reduction does not lose the factor $1/2$.

4. *If $m' > m$ and $m'n > mn'$ then $\mathcal{B}$ has roughly $k = \lceil m'/m \rceil$ times the running time of $\mathcal{A}$ and*

$$\mathrm{Adv}^{(m',n')\text{-cpa}}_{\mathsf{KEM}}(\mathcal{B}) \geq \frac{1}{2}\binom{n'}{m'}\binom{\lceil nm'/m \rceil}{m'}^{-1}\left(\mathrm{Adv}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{A})\right)^k \ .$$

An analogous statement holds between (m, n)-CCA and (m', n')-CCA. If $\mathcal{A}$ queries its decryption oracle q times, then adversary $\mathcal{B}$ queries its decryption oracle at most q, q, kq, and kq times respectively.

3.3 Scaling Factor

We now formally define the scaling factor of key-encapsulation mechanisms. We fix a computational model that associates to each adversary $\mathcal{A}$ its running time. Let $\mathrm{MinTime}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}$ be the minimal time T for which there exists an adversary $\mathcal{A}$ that runs in at most time T and achieves overwhelming advantage $\mathrm{Adv}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{A})$.

We define the scaling factor of KEM relative to (m, n)-CPA security as

$$\mathrm{SF}^{(m,n)\text{-cpa}}_{\mathsf{KEM}} := \frac{\mathrm{MinTime}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}}{\mathrm{MinTime}^{(1,1)\text{-cpa}}_{\mathsf{KEM}}} \ .$$

The scaling factor of KEM relative to (m, n)-CCA security, $\mathrm{SF}^{(m,n)\text{-cca}}_{\mathsf{KEM}}$, is defined in the same way relative to advantage $\mathrm{Adv}^{(m,n)\text{-cca}}_{\mathsf{KEM}}(\mathcal{A})$. By the results of Section 3.2 we can give concrete bounds on the scaling factor (which also hold in the CCA setting).

$$\mathrm{SF}^{(m,n)\text{-cpa}}_{\mathsf{KEM}} \leq \mathrm{SF}^{(m,m)\text{-cpa}}_{\mathsf{KEM}} \leq m \ .$$

The lower bound follows since any adversary against (m, m)-CPA is also an adversary against (m, n)-CPA with the same advantage (Theorem 1, item 1). The upper bound follows from Theorem 1, item 3.

Surprisingly, the scaling factor can be smaller than 1: Being able to chose which users to attack can make the task of breaking multiple instances easier than breaking a single one. An artificial example of a KEM with scaling factor of m/n is sketched in Appendix C. This is, however, a phenomenon limited to the case $m \neq n$: For $n = m$, we know that $\mathrm{SF}_{\mathsf{KEM}}^{(n,m)\text{-cpa}} \geq 1$ by Theorem 1, item 1. Importantly, specific KEMs such as HEG or Cramer-Shoup are known to be "random self-reducible", which implies $\mathrm{MinTime}_{\mathsf{KEM}}^{(1,n)\text{-cpa}} = \mathrm{MinTime}_{\mathsf{KEM}}^{(1,1)\text{-cpa}}$, and hence by Theorem 1, item 1:

$$1 \leq \mathrm{SF}_{\mathsf{KEM}}^{(m,n)\text{-cpa}} \leq m \ .$$

Remark 1. Our definition of scaling factor exclusively considers adversaries achieving overwhelming advantages. This definition is generalized naturally to encompass adversaries with arbitrary advantage as follows. Let $\mathrm{MinTime}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}(\varepsilon)$, associated with $0 \leq \varepsilon \leq 1$, denote the running time of the fastest adversary achieving advantage at least ε in game (m,n)-CPA. Intuitively, an optimally scaling scheme requires m independent execution of a $(1,1)$-CPA adversary in order to break m instances of the scheme. Hence, an advantage-dependent scaling factor for advantage ε is defined as $\mathrm{SF}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}(\varepsilon) := \mathrm{MinTime}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}(\varepsilon^m)/\mathrm{MinTime}_{\mathsf{KEM}}^{(1,1)\text{-cpa}}(\varepsilon)$. Our definition of scaling factor is a special case of this generalized definition for overwhelming success probability. Again, we can use Theorem 1 to show that, for every $0 \leq \varepsilon \leq 1$, $\mathrm{SF}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}(\varepsilon) \leq \mathrm{SF}_{\mathsf{KEM}}^{(m,m)\text{-cpa}}(\varepsilon) \leq m$.

3.4 Multi-Instance Diffie-Hellman-Type Problems

Gap Discrete Logarithm Problem. The m-out-of-n multi-instance gap discrete logarithm problem ((m,n)-GapDL) requires to find the discrete logarithms of at least m out of n input group elements given access to a decisional Diffie-Hellman oracle. We consider three variants of the problem, which differ in their granularity. For high granularity all discrete logarithm challenges are sampled with respect to a fixed group and group generator, while for medium granularity the challenges are elements of a fixed group but defined with respect to different group generators. Finally, in the case of low granularity a fresh group and generator is used for each challenge.

Formally, let $m, n \in \mathbb{N}$ such that $1 \leq m \leq n$ and consider game $\mathrm{G}_{\mathsf{GGen},\mathtt{gran}}^{(m,n)\text{-gdl}}(\mathcal{A})$ of Fig. 3 associated to adversary $\mathcal{A}$, group-generating algorithm GGen, and granularity $\mathtt{gran} \in \{\mathtt{high}, \mathtt{med}, \mathtt{low}\}$. In the game, a vector $\boldsymbol{\mathcal{G}}$ of n group descriptions is set up according to the desired level of granularity using parameter generation algorithm $\mathsf{PGen}[\mathtt{gran}]$. Each entry of $\boldsymbol{\mathcal{G}}$ is of the form $(\mathbb{G}, p, g)$ with $\mathbb{G}$ being a group of prime order p generated by g. After the setup of $\boldsymbol{\mathcal{G}}$ the three variants of the game proceed in the same way. A vector $\boldsymbol{x}$ of length n is sampled, where $\boldsymbol{x}[i]$ is uniformly distributed in $\mathbb{Z}_{\boldsymbol{p}[i]}$. The corresponding challenge vector contains the group elements $\boldsymbol{X}[i] = \boldsymbol{g}[i]^{\boldsymbol{x}[i]}$. At the end of the game, adversary $\mathcal{A}$ outputs a list of indices $L \subseteq [1\,..\,n]$ and a vector $\boldsymbol{x'}$ of length n, where the i-th entry is in $\mathbb{Z}_{\boldsymbol{p}[i]}$. The adversary wins if L contains at least m elements and if the vector $\boldsymbol{x'}$ coincides with $\boldsymbol{x}$ for all indices in L. Additionally, the adversary has access to an oracle DDH, which, on input of index $i \in [1\,..\,n]$ and three group elements $\hat{X}, \hat{Y}, \hat{Z}$, behaves as follows. The game computes the discrete logarithms $\hat{x}, \hat{y}$ of input $\hat{X}, \hat{Y}$ with respect to generator $\boldsymbol{g}[i]$, and then returns 1 if and only if $\boldsymbol{g}[i]^{\hat{x}\hat{y}} = \hat{Z}$.

We define $\mathcal{A}$'s advantage in game $\mathrm{G}_{\mathsf{GGen},\mathtt{gran}}^{(m,n)\text{-gdl}}(\mathcal{A})$ as

$$\mathrm{Adv}_{\mathsf{GGen},\mathtt{gran}}^{(m,n)\text{-gdl}}(\mathcal{A}) = \Pr[\mathrm{G}_{\mathsf{GGen},\mathtt{gran}}^{(m,n)\text{-gdl}}(\mathcal{A})] \ .$$

The m-out-of-n multi-instance discrete logarithm ((m,n)-DL) problem is defined as (m,n)-GapDL with the restriction that $\mathcal{A}$ cannot query DDH.

Gap Computational Diffie-Hellman Problem. The m-out-of-n multi-instance gap computational Diffie-Hellman problem ((m,n)-GapCDH) requires, on input of vectors $g^{\boldsymbol{x}}$ and $g^{\boldsymbol{y}}$, to compute at least m elements of the form $g^{\boldsymbol{x}[i]\boldsymbol{y}[i]}$ for distinct $i \in [1\,..\,n]$. As in the corresponding DL game, the adversary has access to an oracle DDH which computes whether three given group elements are a Diffie-Hellman triple. As in the definition of (m,n)-GapDL, we consider three variants of the problem, which differ in their granularity.

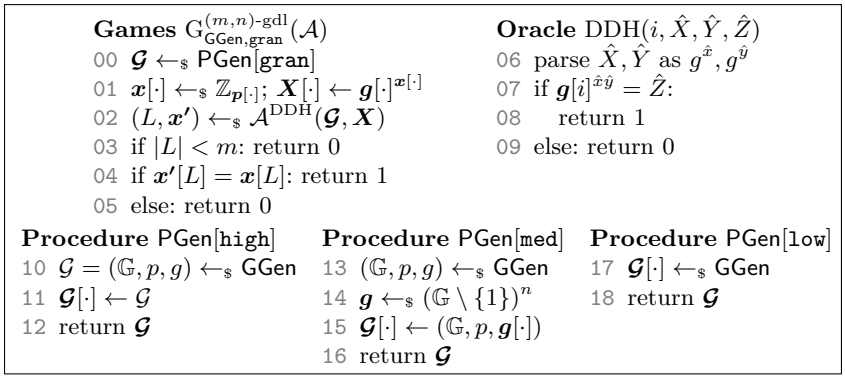

Fig. 3. Security game $\mathrm{G}^{(m,n)\text{-gdl}}_{\mathsf{GGen},\mathtt{gran}}(\mathcal{A})$ for $\mathtt{gran} \in \{\mathtt{high}, \mathtt{med}, \mathtt{low}\}$ modeling the m-out-of-n multi-instance discrete logarithm problem and the gap version.

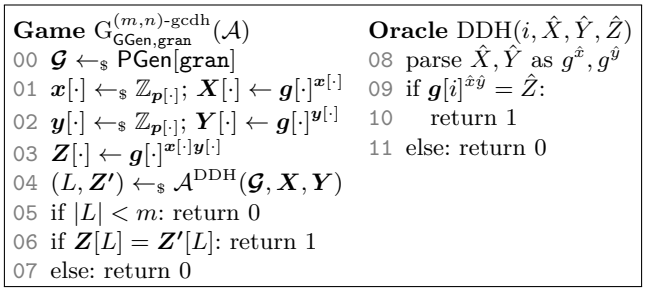

Fig. 4. Security game $\mathrm{G}^{(m,n)\text{-gcdh}}_{\mathsf{GGen},\mathtt{gran}}(\mathcal{A})$ for $\mathtt{gran} \in \{\mathtt{high}, \mathtt{med}, \mathtt{low}\}$ modeling the m-out-of-n multi-instance gap computational Diffie-Hellman problem. PGen is defined in Fig. 3.

Formally, for $m, n \in \mathbb{N}$ s.t. $1 \leq m \leq n$ and consider game $\mathrm{G}^{(m,n)\text{-gcdh}}_{\mathsf{GGen},\mathtt{gran}}(\mathcal{A})$ of Fig. 4 associated to adversary $\mathcal{A}$, group-generating algorithm GGen, and granularity $\mathtt{gran} \in \{\mathtt{high}, \mathtt{med}, \mathtt{low}\}$. In the game, a vector $\boldsymbol{\mathcal{G}}$ of n group descriptions is set up according to parameter generation algorithm $\mathsf{PGen}[\mathtt{gran}]$. After the setup of $\boldsymbol{\mathcal{G}}$ the three variants of the game proceed in the same way. Two vectors $\boldsymbol{x}$, $\boldsymbol{y}$ of length n are sampled, where $\boldsymbol{x}[i]$, $\boldsymbol{y}[i]$ are uniformly distributed in $\mathbb{Z}_{\boldsymbol{p}[i]}$. The corresponding challenge vectors contain the group elements $\boldsymbol{X}[i] = \boldsymbol{g}[i]^{\boldsymbol{x}[i]}$ and $\boldsymbol{Y}[i] = \boldsymbol{g}[i]^{\boldsymbol{y}[i]}$. Additionally, the adversary has access to an oracle DDH, which behaves as described for $\mathrm{G}^{(m,n)\text{-gdl}}_{\mathsf{GGen},\mathtt{gran}}(\mathcal{A})$. At the end of the game, adversary $\mathcal{A}$ outputs a list of indices $L \subseteq [1\,..\,n]$ and a vector $\boldsymbol{Z'}$ of length n, where the i-th entry is an element of the group represented by $\boldsymbol{\mathcal{G}}[i]$. The adversary wins if L contains at least m elements and if the vector $\boldsymbol{Z'}$ coincides with $\boldsymbol{Z}$ for all indices in L.

We define $\mathcal{A}$'s advantage in game $\mathrm{G}^{(m,n)\text{-gcdh}}_{\mathsf{GGen},\mathtt{gran}}(\mathcal{A})$ as

$$\mathrm{Adv}^{(m,n)\text{-gcdh}}_{\mathsf{GGen},\mathtt{gran}}(\mathcal{A}) = \Pr[\mathrm{G}^{(m,n)\text{-gcdh}}_{\mathsf{GGen},\mathtt{gran}}(\mathcal{A})] \ .$$

Further, the m-out-of-n multi-instance computational Diffie-Hellman ((m,n)-CDH) problem is defined as (m,n)-GapCDH with the restriction that $\mathcal{A}$ cannot query oracle DDH.

4 Hashed ElGamal in the Multi-Instance Setting

We investigate the multi-instance security of the well-known Hashed-ElGamal key-encapsulation mechanism [1]. We consider three variants, $\mathsf{HEG}[\mathsf{GGen}, \mathtt{high}]$, $\mathsf{HEG}[\mathsf{GGen}, \mathtt{med}]$, and $\mathsf{HEG}[\mathsf{GGen}, \mathtt{low}]$, corresponding to high, medium, and low granularity respectively. After giving formal definitions of these variants in Section 4.1, in Section 4.2 we prove the main result of this section: The multi-instance security of each variant of the KEM in the random oracle model is tightly implied by the hardness of (m,n)-GapCDH$[\mathsf{GGen}, \mathtt{gran}]$ for the corresponding granularity. Finally, in Section 4.3 we compute lower bounds on the scaling factor of $\mathsf{HEG}[\mathsf{GGen}, \mathtt{gran}]$ for $\mathsf{GGen} \in \{\mathsf{GGen}_{\mathbb{F}_\ell^*}, \mathsf{GGen}_{\mathbb{E}(\mathbb{F}_\ell)}\}$ and $\mathtt{gran} \in \{\mathtt{high}, \mathtt{med}, \mathtt{low}\}$.

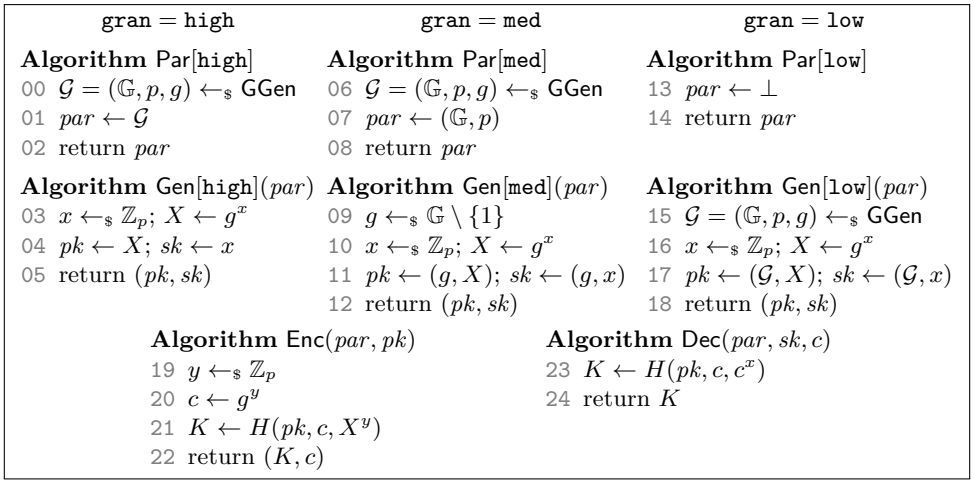

Fig. 5. Variants of Hashed-ElGamal KEM HEG[GGen, high], HEG[GGen, med], and HEG[GGen, low] relative to hash function H and group-generating algorithm GGen. The KEMs share the same encapsulation and decapsulation algorithms. Note that both (par, pk) or (par, sk) determine group description $(\mathbb{G}, p, g)$ and key pk.

4.1 Hashed-ElGamal Key Encapsulation

We consider three variants of the Hashed-ElGamal KEM, defined relative to a hash function H and differing in the way parameters and key-pairs are generated. For high granularity the parameters specify a group description $\mathcal{G} = (\mathbb{G}, p, g)$ with a fixed generator g. Key-pairs (pk, sk) are of the form $pk = X = g^x$ and $sk = x$, where x is randomly sampled in $\mathbb{Z}_p$. For medium granularity the parameters consist of a group $\mathbb{G}$ of order p, but no fixed generator. In this case $pk = (g, g^x)$ and $sk = (g, x)$, where g is a randomly chosen generator of the group $\mathbb{G}$. Finally, for low granularity empty parameters are used. Correspondingly, in this case public keys are of the form $pk = (\mathcal{G}, g^x)$ and secret keys of the form $sk = (\mathcal{G}, x)$, where $\mathcal{G} = (\mathbb{G}, p, g)$ is a freshly sampled group description.

Note that in all three cases the parameters par and a key-pair (pk, sk) generated with respect to par determine a group description $(\mathbb{G}, p, g)$ as well as x and X. In all three variants encapsulated keys are of the form $H(pk, g^y, X^y)$ with corresponding ciphertext g^y, where the y is sampled at random in $\mathbb{Z}_p$. The decapsulation of a ciphertext c is given by $H(pk, c, c^x)$. A formal description of the algorithms describing the Hashed-ElGamal key-encapsulation mechanism for each of the three considered variants can be found in Fig. 5.

4.2 Multi-Instance Security of Hashed ElGamal

The following theorem shows that the security against chosen-ciphertext attacks of each variant of HEG in the multi-instance setting tightly reduces to the corresponding (m, n)-GapCDH problem[3]. Its proof is a generalization of the single-instance version [1] and can be found in Appendix D.

Theorem 2. *Let $m, n \in \mathbb{N}$ with $1 \leq m \leq n$, let $\mathtt{gran} \in \{\mathtt{high}, \mathtt{med}, \mathtt{low}\}$, let GGen be a group-generating algorithm, and let HEG[GGen, gran] be the Hashed-ElGamal KEM of Fig. 5 relative to hash function H. If H is modeled as a random oracle and if the (m, n)-GapCDH[GGen, gran] problem is hard, then HEG[GGen, gran] is (m, n)-CCA secure. Formally, for every adversary $\mathcal{A}$ against game $\mathrm{G}^{(m,n)\text{-cca}}_{\mathsf{HEG}[\mathsf{GGen},\mathtt{gran}]}$ making at most q queries to random oracle RO there exists an adversary $\mathcal{B}$ against game $\mathrm{G}^{(m,n)\text{-gcdh}}_{\mathsf{GGen},\mathtt{gran}}$ that makes at most q queries to DDH and runs in essentially the same time as $\mathcal{A}$ and satisfies*

$$\mathrm{Adv}^{(m,n)\text{-gcdh}}_{\mathsf{GGen},\mathtt{gran}}(\mathcal{B}) \geq \mathrm{Adv}^{(m,n)\text{-cca}}_{\mathsf{HEG}[\mathsf{GGen},\mathtt{gran}]}(\mathcal{A}) \ .$$

[3] The same result holds under the multi-instance version of the strong Diffie-Hellman assumption [1], a falsifiable assumption that is implied by (m, n)-GapCDH.

4.3 Scaling Factor of Hashed ElGamal for Different Parameters

Below we compute the scaling factor of Hashed-ElGamal key encapsulation for different parameter choices. Recall that the scaling factor is given by

$$\mathrm{SF}^{(m,n)\text{-cca}}_{\mathsf{HEG}[\mathsf{GGen},\mathtt{gran}]} = \mathrm{MinTime}^{(m,n)\text{-cca}}_{\mathsf{HEG}[\mathsf{GGen},\mathtt{gran}]} / \mathrm{MinTime}^{(1,1)\text{-cca}}_{\mathsf{HEG}[\mathsf{GGen},\mathtt{gran}]} \ .$$

Note, that the multi-instance security of HEG can be broken by computing m public keys, which corresponds to computing m DL instances. On the other hand, from Theorem 2 we know that the (m,n)-CCA-security of HEG is tightly implied by (m,n)-GapCDH. Thus,

$$\mathrm{MinTime}^{(m,n)\text{-gcdh}}_{\mathsf{GGen},\mathtt{gran}} \leq \mathrm{MinTime}^{(m,n)\text{-cca}}_{\mathsf{HEG}[\mathsf{GGen},\mathtt{gran}]} \leq \mathrm{MinTime}^{(m,n)\text{-dl}}_{\mathsf{GGen},\mathtt{gran}} \ .$$

Hence, we can bound the scaling factor of Hashed ElGamal as

$$\mathrm{SF}^{(m,n)\text{-cca}}_{\mathsf{HEG}[\mathsf{GGen},\mathtt{gran}]} \geq \mathrm{MinTime}^{(m,n)\text{-gcdh}}_{\mathsf{GGen},\mathtt{gran}} / \mathrm{MinTime}^{(1,1)\text{-dl}}_{\mathsf{GGen},\mathtt{gran}} \ .$$

Below we consider two instantiations of group-generating algorithms: $\mathsf{GGen}_{\mathbb{F}^*_\ell}$ and $\mathsf{GGen}_{\mathbb{E}(\mathbb{F}_\ell)}$. Due to either Hypothesis 1 from the introduction or the results of Sections 5 and 6 respectively, for both instantiations solving (m,n)-GapCDH is as hard as (m,n)-GapDL. Thus, the lower bounds on the scaling factor derived below are sharp.

Hashed ElGamal in the Finite-Field Setting. Assuming the correctness of Hypothesis 1, we conclude that $\mathrm{MinTime}^{(m,n)\text{-gcdh}}_{\mathbb{F}^*_\ell,\mathtt{gran}} = \mathrm{MinTime}^{(m,n)\text{-dl}}_{\mathbb{F}^*_\ell,\mathtt{gran}}$ is given by

$$\begin{aligned} &L_\ell(1/3, 1.902) + m \cdot L_\ell(1/3, 1.232) \quad \text{for } \mathtt{gran} \in \{\mathtt{high}, \mathtt{med}\} \ , \text{ and} \\ &\min\{m \cdot L_\ell(1/3, 1.902), L_\ell(1/3, 2.007) + m \cdot L_\ell(1/3, 1.639)\} \quad \text{for } \mathtt{gran} = \mathtt{low} \ . \end{aligned}$$

We obtain the scaling factor by dividing by $\mathrm{MinTime}^{(1,1)\text{-dl}}_{\mathbb{F}^*_\ell,\mathtt{gran}} = L_\ell(1/3, 1.902)$. Defining δ via $m = L_\ell(1/3, \delta)$ we can rewrite $m \cdot L_\ell(1/3, 1.232)$ as $L_\ell(1/3, \delta + 1.232)$. For $\delta \leq 0.67$ we get $L_\ell(1/3, 1.902) \geq L_\ell(1/3, \delta + 1.232)$. Hence for these values of δ the scaling factor for medium and high granularity is roughly 1. For larger m, on the other hand, it is of order $L_\ell(1/3, \delta - 0.67)$.

Summing up for $\mathtt{gran} \in \{\mathtt{med}, \mathtt{high}\}$ we obtain

$$\mathrm{SF}^{(m,n)\text{-cca}}_{\mathsf{HEG}[\mathsf{GGen}_{\mathbb{F}^*_\ell},\mathtt{gran}]} = \begin{cases} 1 & \delta \leq 0.67 \\ L_\ell(1/3, \delta - 0.67) & \delta > 0.67 \end{cases} \ .$$

Further, we get $L_\ell(1/3, \delta + 1.902) \leq L_\ell(1/3, 2.007)$ for $\delta \leq 0.105$. Hence in this case for low granularity the scaling factor is given by $m = L_\ell(1/3, \delta)$. Moreover, we obtain $L_\ell(1/3, \delta + 1.639) = L(1/3, 2.007)$ for $\delta = 0.368$ implying that for $0.108 \leq \delta \leq 0.368$ the scaling factor is of order $L_\ell(1/3, 2.007 - 1.902)$ and of order $L_\ell(1/3, \delta + 1.639 - 1.902)$ for larger values of δ. Summing up:

$$\mathrm{SF}^{(m,n)\text{-cca}}_{\mathsf{HEG}[\mathsf{GGen}_{\mathbb{F}^*_\ell},\mathtt{low}]} = \begin{cases} L_\ell(1/3, \delta) & 0 \leq \delta < 0.105 \\ L_\ell(1/3, 0.105) & 0.105 \leq \delta < 0.368 \\ L_\ell(1/3, -0.263 + \delta) & 0.368 \leq \delta \end{cases} \ .$$

Formally, the asymptotic behavior of the scaling factor computed above is linear[4] in m and hence, at first glance, seems optimal. However, as discussed in the introduction, the numbers of $L_\ell(1/3, 0.67)$ or $L_\ell(1/3, 0.368)$ instances starting from which the cumulative cost of breaking the instances outweighs the cost of the precomputation are typically large.

Hashed ElGamal in the Elliptic-Curve Setting. Recall that $\mathsf{GGen}_{\mathbb{E}(\mathbb{F}_\ell)}$ generates elliptic curves of size $p \approx \ell$ defined over the field $\mathbb{F}_\ell$ for randomly chosen ℓ. If we model elliptic curves as generic groups we can derive the scaling factor as follows. Ignoring constants, a single DL instance can be solved in

[4] For fixed ℓ and very large values of m and n generic attacks start to outperform the NFS and the scaling factor actually becomes $\Theta(\sqrt{m})$.

time $O(\sqrt{p})$. The lower bounds derived in Section 6 (Corollaries 3 and 4 and Theorem 6) imply the following: A generic algorithm solving (m,n)-GapCDH for high and medium granularity performs at least $\Omega(\sqrt{mp})$ group operations; the low-granularity case requires at least $\Omega(m\sqrt{p})$ group operations. (In the low-granularity case we formally consider n groups of differing group orders $p_1,\ldots,p_n$. Here p denotes a lower bound on all p_i.) Summing up, we obtain

$$\mathrm{SF}^{(m,n)\text{-cca}}_{\mathsf{HEG}[\mathsf{GGen}_{\mathbb{E}(\mathbb{F}_\ell)},\mathtt{gran}]} = \begin{cases} \Theta(\sqrt{mp}/\sqrt{p}) = \Theta(\sqrt{m}) & \mathtt{gran} \in \{\mathtt{high},\mathtt{med}\} \\ \Theta(m\sqrt{p}/\sqrt{p}) = \Theta(m) & \mathtt{gran} = \mathtt{low} \end{cases} .$$

(The constants hidden within the Θ notation can be made explicit from our results, and are between 0.1 and 6.6.) In Appendix E we illustrate how the scaling factors computed above could be taken into account when choosing parameters for HEG.

5 Generic Hardness of the Multi-Instance Gap Discrete Logarithm Problem

In this section we define a new hard problem, namely the polycheck discrete logarithm problem (PolyDL), in the multi-instance setting. Then, we proceed to show a concrete bound on its security in the generic group model (Theorem 3). Most notably, from this bound we present a concrete bound on the security of GapDL. To prove the bound we define an additional problem, the *search-by-hypersurface* problem (SHS).

In Section 5.1 we define the PolyDL and SHS problems. Next, we show our bounds on the security of GapDL, and further argue that all derived bounds are optimal. Section 5.2 covers high, Section 5.3 medium, and Section 5.4 low granularity.

5.1 Polycheck Discrete Logarithm and Search-by-Hypersurface Problem

Polycheck Discrete Logarithm Problem. The m-out-of-n multi-instance polycheck discrete logarithm problem ((m,n)-d-PolyDL) for polynomials of degree at most d requires to find the discrete logarithms of at least m out of n input group elements given access to a decisional oracle Eval with behaves as follows. Eval takes as input a polynomial $f \in \mathbb{Z}_p[X_1,\ldots,X_k]$ of degree at most d and a list of group elements $(g^{\hat{x}_1},\ldots,g^{\hat{x}_k})$, where k is an arbitrary integer, and returns 1 if and only if $g^{f(\hat{x}_1,\ldots,\hat{x}_k)} = 1$. As usual, we consider three variants of the problem, which differ in their granularity.

Formally, let $m,n,d \in \mathbb{N}$ such that $1 \le m \le n$, $d \ge 1$, and consider game $\mathrm{G}^{(m,n)\text{-}d\text{-polydl}}_{\mathsf{GGen},\mathtt{gran}}(\mathcal{A})$ of Fig. 6 associated to adversary $\mathcal{A}$ and granularity $\mathtt{gran} \in \{\mathtt{high},\mathtt{med},\mathtt{low}\}$. In the game, a vector $\boldsymbol{\mathcal{G}}$ of n group descriptions is set up according to the desired level of granularity using $\mathsf{PGen}[\mathtt{gran}]$. After the setup of $\boldsymbol{\mathcal{G}}$ the three variants of the game proceed in the same way. A vector $\boldsymbol{x}$ of length n is sampled, where $\boldsymbol{x}[i]$ is uniformly distributed in $\mathbb{Z}_{\boldsymbol{p}[i]}$. The corresponding challenge vector contains the group elements $\boldsymbol{X}[i] = \boldsymbol{g}[i]^{\boldsymbol{x}[i]}$. At the end of the game, adversary $\mathcal{A}$ outputs a list of indices $L \subseteq [1\,..\,n]$ and a vector $\boldsymbol{x'}$ of length n, where the i-th entry is in $\mathbb{Z}_{\boldsymbol{p}[i]}$. The adversary wins if L contains at least m elements and if the vector $\boldsymbol{x'}$ coincides with $\boldsymbol{x}$ for all indices in L. Additionally, the adversary has access to an evaluation oracle Eval, which on input of an index $i \in [1\,..\,n]$, a polynomial $f \in \mathbb{Z}_p[X_1,\ldots,X_k]$, and a list of group elements $(\hat{X}_1,\ldots,\hat{X}_k)$, where k is an arbitrary integer which might be different on different calls, behaves as follows. If $\deg f > d$, then Eval returns 0. Otherwise, the game computes the discrete logarithms $\boldsymbol{x_i}$ of the challenges $\boldsymbol{X}$ with respect to generator $\boldsymbol{g}[i]$, and then returns 1 if and only if $g[i]^{f(\boldsymbol{x_i})} = 1$.

We define the advantage of $\mathcal{A}$ in game $\mathrm{G}^{(m,n)\text{-}d\text{-polydl}}_{\mathsf{GGen},\mathtt{gran}}(\mathcal{A})$ as

$$\mathrm{Adv}^{(m,n)\text{-}d\text{-polydl}}_{\mathsf{GGen},\mathtt{gran}}(\mathcal{A}) = \Pr[\mathrm{G}^{(m,n)\text{-}d\text{-polydl}}_{\mathsf{GGen},\mathtt{gran}}(\mathcal{A})] .$$

The next definition extends the search-by-hyperplane-query problem (SHQ) by Yun [23].

Search-by-Hypersurface Problem. The search-by-hypersurface problem in dimension n for polynomials of degree at most d (n-SHS$_d$) requires to find a randomly sampled point $\boldsymbol{a}$ of the space by adaptively checking whether point $\boldsymbol{a}$ is contained in the queried hypersurface (i.e., the set of zeroes of a polynomial).

Formally, let $n,d,p \in \mathbb{N}$ such that p is prime and $d,n \ge 1$, and consider game $\mathrm{G}^{n\text{-shs}_d}_p(\mathcal{A})$ of Fig. 7 associated to adversary $\mathcal{A}$. In the game, a vector $\boldsymbol{a}$ of length n is sampled, where $\boldsymbol{a}[i]$ is uniformly

Game $\mathrm{G}^{(m,n)\text{-}d\text{-polydl}}_{\mathsf{GGen},\mathtt{gran}}(\mathcal{A})$	**Oracle** $\mathrm{Eval}(i, f, \hat{\boldsymbol{X}})$
00 $\boldsymbol{\mathcal{G}} \leftarrow_{\$} \mathsf{PGen}[\mathtt{gran}]$	06 if $\deg f > d$: return 0
01 $\boldsymbol{x}[\cdot] \leftarrow_{\$} \mathbb{Z}_{\boldsymbol{p}[\cdot]}$; $\boldsymbol{X}[\cdot] \leftarrow \boldsymbol{g}[\cdot]^{\boldsymbol{x}[\cdot]}$	07 parse $\hat{\boldsymbol{X}}$ as $\boldsymbol{g}[i]^{\hat{\boldsymbol{x}}}$
02 $(L, \boldsymbol{x'}) \leftarrow_{\$} \mathcal{A}^{\mathrm{Eval}}(\boldsymbol{\mathcal{G}}, \boldsymbol{X})$	08 if $\boldsymbol{g}[i]^{f(\hat{\boldsymbol{x}})} = 1$:
03 if $\lvert L\rvert < m$: return 0	09 return 1
04 if $\boldsymbol{x'}[L] = \boldsymbol{x}[L]$: return 1	10 else: return 0
05 else: return 0	

Fig. 6. Security game $\mathrm{G}^{(m,n)\text{-}d\text{-polydl}}_{\mathsf{GGen},\mathtt{gran}}(\mathcal{A})$ relative to $\mathsf{GGen}, \mathtt{gran}$, modeling the m-out-of-n multi-instance polycheck discrete logarithm problem for polynomials of degree at most d. We assume that polynomial f input to Eval has $\lvert\hat{\boldsymbol{X}}\rvert$ indeterminates. PGen is defined in Fig. 3.

Game $\mathrm{G}^{n\text{-shs}_d}_p(\mathcal{A})$	**Oracle** $\mathrm{Eval}(f)$
00 $\boldsymbol{a} \leftarrow_{\$} \mathbb{Z}_p^n$	04 if $\deg(f) > d$: return 0
01 $\boldsymbol{a'} \leftarrow_{\$} \mathcal{A}^{\mathrm{Eval}}(p)$	05 if $f(\boldsymbol{a}) = 0$: return 1
02 if $\boldsymbol{a'} = \boldsymbol{a}$: return 1	06 else: return 0
03 else: return 0	

Fig. 7. Security game $\mathrm{G}^{n\text{-shs}_d}_p(\mathcal{A})$ with respect to integer d and prime p modeling the search-by-hypersurface problem on dimension n for polynomials of degree at most d. All inputs f to oracle Eval are elements of the polynomial ring $\mathbb{Z}_p[X_1, \ldots, X_n]$.

distributed in $\mathbb{Z}_p$. At the end of the game, adversary $\mathcal{A}$ outputs a vector $\boldsymbol{a'} \in \mathbb{Z}_p^n$. The adversary wins if $\boldsymbol{a'} = \boldsymbol{a}$. Additionally, the adversary has access to an evaluation oracle Eval, which on input of a polynomial $f \in \mathbb{Z}_p[X_1, \ldots, X_n]$ behaves as follows. If $\deg f > d$, then Eval returns 0. Otherwise, the oracle returns 1 if and only if $f(\boldsymbol{a}) = 0$.

We define the advantage of $\mathcal{A}$ in game $\mathrm{G}^{n\text{-shs}_d}_p(\mathcal{A})$ as

$$\mathrm{Adv}^{n\text{-shs}_d}_p(\mathcal{A}) = \Pr[\mathrm{G}^{n\text{-shs}_d}_p(\mathcal{A})] \ .$$

5.2 Generic Hardness of High-Granularity (m, n)-d-PolyDL

Below, we state the main result of this section, an explicit upper bound on the security of high-granularity (n, n)-d-PolyDL in the generic group model.

Note that this bound is of particular interest in the context of generic bilinear (or even multilinear) maps. In fact, a d-linear map yields a natural way to compute any answer of oracle Eval for polynomials of degree at most d in the base group.

Theorem 3. *Let n, d be positive integers and p a prime number. Let $\mathsf{GGen}_{\mathsf{gg}}$ be a group-generating algorithm that generates generic groups of exactly size p. Then for every generic adversary $\mathcal{A}$ against (n, n)-d-PolyDL$[\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}]$ that makes at most q queries to the group-operation oracle and q_{Eval} queries to oracle* Eval*:*

$$\mathrm{Adv}^{(n,n)\text{-}d\text{-polydl}}_{\mathsf{GGen}_{\mathsf{gg}},\mathtt{high}}(\mathcal{A}) \leq \left(\frac{d}{p}\right)^n + \frac{1}{2}\left(\frac{ed(q+n+1)^2 + 2edq_{\mathrm{Eval}}}{2np}\right)^n \ .$$

This extends [23, Corollary 2] from standard DL to the polycheck case. Most importantly, it allows us to prove the following theorem.

Corollary 1. *Let n be any positive integer and $\mathsf{GGen}_{\mathsf{gg}}$ be a group-generating algorithm that generates generic groups of at least size p. Then for every generic adversary $\mathcal{A}$ against (n, n)-*GapDL$[\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}]$ *that makes at most q queries to the group-operation oracle and q_{DDH} queries to the DDH oracle:*

$$\mathrm{Adv}^{(n,n)\text{-gdl}}_{\mathsf{GGen}_{\mathsf{gg}},\mathtt{high}}(\mathcal{A}) \leq \left(\frac{2}{p}\right)^n + \frac{1}{2}\left(\frac{e(q+n+1)^2 + 2eq_{\mathrm{DDH}}}{np}\right)^n \approx \left(\frac{q^2}{np}\right)^n \ .$$

Proof (Corollary 1). Note that oracle DDH of game (n, n)-GapDL can be simulated using oracle Eval from game (n, n)-2-PolyDL. In fact, $g^{xy} = g^z$ if and only if $g^{f(x,y,z)} = 1$, with $f(X_1, X_2, X_3) \coloneqq X_1X_2 - X_3$. Then apply Theorem 3 with $d = 2$. □

The latter result gives a lower bound on the amount of queries required to solve (n, n)-GapDL with overwhelming success probability. In fact, in this case we get $e(q+n+1)^2+2eq_{\mathrm{DDH}} > np$. Assuming p large compared to n and q_{DDH} (which allows us to ignore the additive terms $n+1$ and q_{DDH}), we obtain that the fastest adversary with overwhelming success probability in game (n, n)-GapDL requires $q > \sqrt{np/e}$ group operations. Moreover, there exists a generic algorithm which solves (n, n)-GapDL[$\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}$] with probability 1 in $q < 2\sqrt{np}$ steps (see Appendix F for details). This shows that our result is optimal in terms of number of group operations up to a factor of $2\sqrt{e}$.

The proof of Theorem 3 follows a structure similar to Yun [23]. First we prove the equivalence of n-SHS$_d[p]$ and (n, n)-d-PolyDL[$\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}$], and then we bound the success probability of an adversary against n-SHS$_d[p]$. The equivalence of the two problems corresponds to the lemma below.

Statement and proof closely follow [23, Theorem 1] while additionally handling Eval queries.

Lemma 2. *Let n, d be positive integers and p a prime number. Let $\mathsf{GGen}_{\mathsf{gg}}$ be a group-generating algorithm that generates generic groups of exactly size p. Then for every adversary $\mathcal{A}$ against game (n, n)-d-PolyDL[$\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}$] there exists an adversary $\mathcal{B}$ against n-SHS$_d[p]$ such that*

$$\mathrm{Adv}_p^{n\text{-shs}_d}(\mathcal{B}) \geq \mathrm{Adv}^{(n,n)\text{-}d\text{-polydl}}_{\mathsf{GGen}_{\mathsf{gg}},\mathtt{high}}(\mathcal{A}) \ .$$

Moreover, if $\mathcal{A}$ makes q group-operation queries and q_{Eval} queries to Eval*, then $\mathcal{B}$ makes at most $q_{\mathrm{Eval}} + (n+q)(n+q+1)/2$ queries to* Eval*.*

Proof. Let $\mathcal{A}$ be an adversary against (n, n)-d-PolyDL[$\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}$]. We build adversary $\mathcal{B}$ as follows. Recall that $\mathcal{B}$ can query an oracle Eval on input of polynomials $f \in \mathbb{Z}_p[X_1, \ldots, X_n]$ that returns 1 if $f(\boldsymbol{a}) = 0$ and 0 otherwise, where $\boldsymbol{a}$ is uniformly sampled in $\mathbb{Z}_p^n$ at the start of the game. The prime p, that $\mathcal{B}$ receives as input, serves as the order the generic group. Note that it is correctly distributed.

We introduce the two lists $S = \{s_i\}$ and $L = \{l_i\}$, held by $\mathcal{B}$. The first stores the representation strings of each generic group elements returned to $\mathcal{A}$; the second stores polynomials of degree at most 1 in $\mathbb{Z}_p[X_1, \ldots, X_n]$. $\mathcal{B}$ samples a uniform representation s_0 that serves as the group generator and sets $l_0 = 1$. Then $\mathcal{B}$ defines the i-th DLOG challenge iteratively: First $\mathcal{B}$ sets $l_i = X_i$, then for every $j < i$, $\mathcal{B}$ calls Eval on input $l_i - l_j$. If any query returns 1, $\mathcal{B}$ sets $s_i = s_j$, otherwise $\mathcal{B}$ sets s_i to be a uniform representation among those not already present in S. The group generator is defined as $g = s_0$ and the challenge vector as $\boldsymbol{X} = (s_1, \ldots, s_n)$. Adversary $\mathcal{A}$ is run on input p, g, and $\boldsymbol{X}$.

In the following description, let c be a counter variable updated by $\mathcal{B}$ representing the number of elements of S and L.

Assume that $\mathcal{A}$ queries the group-operation oracle on input $s, s' \in S$. $\mathcal{B}$ retrieves the polynomials $l, l' \in L$ corresponding to s, s' and adds to L the element $l_{c+1} = l + l'$. Then, for every $j < c$ adversary $\mathcal{B}$ calls Eval on input $l_{c+1} - l_j$. If for any j the oracle returns 1 then the adversary adds to S the element $s_{c+1} = s_j$. Otherwise $\mathcal{B}$ sets s_{c+1} to be a uniform representation among those not already present in S. Finally s_{c+1} is returned to $\mathcal{A}$.

Assume that $\mathcal{A}$ queries the oracle Eval on input $f \in \mathbb{Z}_p[X_1, \ldots, X_k], \boldsymbol{s} \in S^k$ for some integer k. If $\deg f > d$, then $\mathcal{B}$ returns 0. Otherwise, $\mathcal{B}$ retrieves the list of polynomials $\boldsymbol{l} \in L^k$, corresponding to each entry of $\boldsymbol{s}$. Then adversary $\mathcal{B}$ calls Eval on input $f(\boldsymbol{l})$ and forwards the output to $\mathcal{A}$.

Finally, when $\mathcal{A}$ terminates $\mathcal{B}$ forwards its output.

We show that $\mathcal{B}$ offers a perfect simulation of the oracle DLOG security game to $\mathcal{A}$. First we observe that the DLOG challenges have been implicitly set to $\boldsymbol{X} = g^{\boldsymbol{a}}$ and each s_i to $g^{l_i(\boldsymbol{a})}$. The initialization of $\boldsymbol{X}$ guarantees that two entries of $\boldsymbol{X}$ coincide if and only if the corresponding entries of $\boldsymbol{a}$ coincide. Similarly, each time a new group element is returned to $\mathcal{A}$, it is checked for equality against all previous elements. Note that all elements known to the adversary are associated to polynomials of degree at most 1 by construction. The oracle Eval is simulated correctly since $g^{f(\boldsymbol{l}(\boldsymbol{a}))} = 1$ if and only if $f(\boldsymbol{l}(\boldsymbol{a})) = 0$. Moreover, $f(\boldsymbol{l})$ is a polynomial of degree at most $\deg f \leq d$ when expanded, since all polynomials in L have degree at most 1.

To conclude, we count the amount of queries to Eval. For every element added to S we need to query Eval as many times as there are elements of S, while each DDH query requires a single query to Eval. The total amount of queries of $\mathcal{B}$ to Eval is then:

$$\sum_{i=0}^{n+q} i + q_{\mathrm{Eval}} = \frac{(n+q)(n+q+1)}{2} + q_{\mathrm{Eval}} \ . \qquad \square$$

We start working on n-SHS$_d[p]$ with the next lemma. Here we express that, up to a loss of d^n, an adversary against n-SHS$_d[p]$ does not need more than n hypersurface queries which return 1 to identify a solution.

Importantly, observe how we limit the resources of an adversary against n-SHS$_d[p]$ exclusively in terms of its queries to Eval. Our adversaries are otherwise unbounded. For this reason, the following reduction does not consider the computational resources needed by the adversary to perform its operations.

Lemma 3. *Let n, d be positive integers and p a prime number. For every adversary $\mathcal{A}$ against n-SHS$_d[p]$ that makes at most q queries to* Eval *there exists an adversary $\mathcal{B}$ against n-SHS$_d[p]$ that makes at most q queries to* Eval *such that at most n of them return 1 and*

$$\mathrm{Adv}_p^{n\text{-shs}_d}(\mathcal{B}) \geq d^{-n}\mathrm{Adv}_p^{n\text{-shs}_d}(\mathcal{A}) \ .$$

Proof Idea. Intuition for the proof is simple for the case $n = 1$: All queries of $\mathcal{A}$ to SimEval are forwarded to Eval. The first time Eval(g) returns 1, we know that the secret $\boldsymbol{a}$ must be a zero of g. Since g has degree at most d, there can be at most d distinct zeroes. The reduction guesses which zero is the correct one (this is the reduction loss) and then simulates the remaining queries of $\mathcal{A}$ to SimEval accordingly. The proof is similar for $n > 1$. We know that, in general, n polynomials in $\mathbb{Z}_p[X_1, \ldots, X_n]$ of degree d have at most d^n zeroes in common, one of which the reduction can use to simulate remaining queries to SimEval. However, the n queried polynomials must be in general position: For example, the zeroes of $x_1 + x_2$ are the same as those of $2x_1 + 2x_2$, and querying both polynomials would not help the reduction. To resolve this issue, the reduction keeps a set Z of common zeroes to all polynomials seen so far which, when forwarded to Eval, make the oracle return 1 (i.e., polynomials which vanish on $\boldsymbol{a}$). This set has a rich structure: In fact, the study of zero sets of polynomial is the raison d'être of the field of algebraic geometry. If the polynomial g queried by $\mathcal{A}$ carries no new information (i.e., $g(Z) = \{0\}$) then the simulated oracle returns 1 without forwarding. Otherwise, the polynomial is forwarded. If the answer is 1, then the reduction updates the set Z and then guesses which one of its irreducible components contains $\boldsymbol{a}$, which becomes the updated Z. The identification of irreducible components is made possible by the underlying structure of the set Z. Selecting an irreducible component guarantees that, on a following evaluation query, intersecting the now irreducible Z with another hypersurface not containing Z brings down the dimension of Z by 1. Since the dimension of $\mathbb{Z}_p^n$ is n, we can have at most n such queries. With a careful choice of the guessing probability of each irreducible component, Bézout's theorem ensures that the probability of always making the right guess is again d^{-m}.

In the following paragraph we introduce the notation needed to prove Lemma 3.

Algebraic Geometry. Let K be a field. We denote with $\overline{K}$ the algebraic closure of the field K. The projective space of dimension n over K is represented with $\mathbb{P}^n(K)$. Let $T \subseteq K[X_0, \ldots, X_n]$ a subset of homogeneous polynomials. We define $Z(T) = \{\boldsymbol{a} \in K^n : f(\boldsymbol{a}) = 0 \ \forall f \in T\}$. A (projective) algebraic set A of $\mathbb{P}^n(K)$ is any set that can be described as $Z(T)$ for some T. We work with the Zariski topology over $\mathbb{P}^n(K)$: Closed sets are the algebraic sets on $\mathbb{P}^n(K)$. A (projective) hypersurface S of $\mathbb{P}^n(K)$ is an algebraic set generated by a single homogeneous polynomial f.[5] An irreducible set X of the topological space $\mathbb{P}^n(K)$ is a set which cannot be expressed as the union of two distinct closed subsets. A (projective algebraic) variety V is an irreducible algebraic set. Given an algebraic set A, we say that $A = V_1 \cup \ldots \cup V_k$ is an irreducible decomposition (and V_i an irreducible component) if all sets V_i are varieties and $V_i \nsubseteq V_j$ for $i \neq j$. The dimension of a set $X \subseteq \mathbb{P}^n(K)$ ($\dim X$) is the supremum of all integers m such that there exists a chain $V_0 \subsetneq \ldots \subsetneq V_m$ of distinct irreducible closed subsets of X. For a definition of the degree of an algebraic set A ($\deg A$) we refer to [15, page 52].

Proof (Lemma 3). Let $\mathcal{A}$ be an adversary against n-SHS$_d[p]$ that makes at most q queries to SimEval. Without loss of generality, we assume $\mathcal{A}$ only queries polynomials of degree between 1 and d.

We build adversary $\mathcal{B}$ as follows. $\mathcal{B}$ sets two internal variables: $Z \leftarrow \mathbb{P}^n\big(\overline{\mathbb{Z}_p}\big)$ (the n-dimensional projective space over the algebraic closure of $\mathbb{Z}_p$), and $t \leftarrow 1$. Subsequently, $\mathcal{B}$ runs $\mathcal{A}$, proceeds to simulate each query of $\mathcal{A}$ to SimEval as described below, and eventually forwards its output.

Suppose that $\mathcal{A}$ queries SimEval on input of a polynomial $g' \in \mathbb{Z}_p[X_1, \ldots, X_n]$. Then $\mathcal{B}$ computes the homogenization of g' and stores it as a variable $g \in \mathbb{Z}_p[X_0, \ldots, X_n]$. If $g(Z) = \{0\}$ (all points in Z vanish

[5] This is the only definition that differs from Hartshorne [15], who also requires f be irreducible.

Adversary $\mathcal{B}^{\text{Eval}}$	**Oracle** SimEval(g') $\quad \backslash\backslash\, g' \in \mathbb{Z}_p[X_1, \ldots, X_n],\ 1 \le \deg g \le d$
00 $Z \leftarrow \mathbb{P}^n(\overline{\mathbb{Z}_p})$	04 $g \leftarrow$ homogenize(g')
01 $t \leftarrow 1$	05 if $g(Z) = \{0\}$: return 1
02 $\boldsymbol{a} \leftarrow_{\$} \mathcal{A}^{\text{SimEval}}$	06 $Z' \leftarrow \{z \in Z : g(z) = 0\}$
03 return $\boldsymbol{a}$	07 if $Z' = \emptyset$: return 0
	08 $b \leftarrow \text{Eval}(g')$
	09 if $b = 1$:
	10 $\quad V_1 \cup \ldots \cup V_l \leftarrow Z'$
	11 $\quad$ for $i \in [1\,..\,l-1]$:
	12 $\quad\quad p_i \leftarrow \deg V_i/(t \deg g)$
	13 $\quad p_l \leftarrow 1 - p_1 - \ldots - p_{l-1}$
	14 $\quad$ pick j among $[1\,..\,l]$ with distribution $\{p_i\}$
	15 $\quad Z \leftarrow V_j$
	16 $\quad t \leftarrow \deg V_j$
	17 return b

Fig. 8. Adversary $\mathcal{B}$ for the proof of Lemma 3 with respect to adversary $\mathcal{A}$.

on g), then $\mathcal{B}$ returns 1. $\mathcal{B}$ then computes the zeroes of g in Z (equivalently: the intersection of Z with the algebraic set of zeroes of g) and stores the result in a variable Z'. If $Z' = \emptyset$, then $\mathcal{B}$ returns 0. Afterwards, $\mathcal{B}$ forwards the response of its own oracle and updates its internal variables as follows. If the output of the forwarded query is 0, then everything remains unchanged. Otherwise, the adversary decomposes Z' as a union of irreducible varieties of the form $Z' = V_1 \cup \ldots \cup V_l$. $\mathcal{B}$ computes then the degree of each component, $\deg V_i$ for $i \in [1\,..\,l]$. This is used to define a probability distribution over the set $[1\,..\,l]$ as $p_i = \deg V_i/(t \cdot \deg g)$ for $i \in [1\,..\,l-1]$ and $p_l = 1 - p_1 - \ldots - p_{l-1}$. An index j is then picked from $[1\,..\,l]$ according to the distribution defined by the p_i's. Then $\mathcal{B}$ redefines $Z \leftarrow V_j$ and $t \leftarrow \deg V_j$.

Adversary $\mathcal{B}$ is also described in Fig. 8, which is used as reference for conciseness.

First, we show that $\mathcal{B}$ terminates without errors. To this end, we show that (a) there exists a finite decomposition as in line 10, (b) such a decomposition is nonempty, (c) there is no division by zero in line 12, and (d) the reals p_i defined in lines 12 and 13 describe a probability distribution. For (a), the irreducible decomposition exists by [15, Exercise 2.5(b)][6]. For (b), observe that the decomposition of Z' is empty if and only if $Z' = \emptyset$. This does not happen by the check in line 07. For (c), we observe that, by construction, $\deg g = \deg g' \ge 1$. At the start of the proof $t = 1$ and by [15, Proposition 7.6(a)] the degree of a proper variety is always a positive integer. For (d), assume the condition in line 09 is triggered. Since multiplicities are positive integers, Bézout's theorem [15, Theorem 7.7] implies[7] that

$$\sum_{i=1}^{l} \deg V_i \le \deg Z \cdot \deg g \ .$$

By their definitions, $\deg Z = t$ and $p_i = \deg V_i/(t \deg g)$ for $i \in [1\,..\,l-1]$ after running line 12. (Note that at the start we set $t = \deg \mathbb{P}^n(\overline{\mathbb{Z}_p}) = 1$ [15, Proposition 7.6(c)].) We can thus rewrite the previous expression as:

$$\sum_{i=1}^{l-1} p_i + \frac{\deg V_l}{t \deg g} = \sum_{i=1}^{l} \frac{\deg V_i}{t \deg g} \le 1 \ .$$

Since by definition $\deg V_i/(t \deg g) > 0$ for $i \in [1\,..\,l]$, we also get $p_i \le 1$ for $i \in [1\,..\,l-1]$ and

$$1 \ge p_l = 1 - \sum_{i=1}^{l-1} p_i = 1 - \sum_{i=1}^{l-1} \frac{\deg V_i}{t \deg g} \ge \frac{\deg V_l}{t \deg g} > 0 \ . \tag{2}$$

[6] All citations to Hartshorne [15] refer to Chapter I.

[7] Technically, [15, Theorem 7.7] requires intersecting hypersurface H be irreducible, while in our case g might be reducible. This condition is not necessary: If $H = \bigcup H_i$ (irreducible decomposition) then by [15, Theorem 7.2] $\dim H_i \cap H_j = n - 2$, by [15, Proposition 7.6(b)] $\deg H = \sum \deg H_i$, and we can apply [15, Theorem 7.7] on each H_i and sum up the result.

As a next step, we show that at most n queries of adversary $\mathcal{B}$ to Eval are answered with 1. Let m be the number of times the check in line 09 is triggered. We want to prove that $m \leq n$. Let Z_i, Z'_i and g_i be the values assigned to varieties Z, Z' and polynomial g after a full execution of SimEval the i-th time the query to Eval is answered with 1, and set $Z_0 = \mathbb{P}^n\big(\overline{\mathbb{Z}_p}\big)$. We want to prove by induction that $\dim Z_i = n - i$. Then, since by definition the dimension is a nonnegative integer, $n \geq m = \max\{i\}$. By definition $\dim Z_0 = \dim \mathbb{P}^n\big(\overline{\mathbb{Z}_p}\big) = n$ [15, Exercise 2.7(a)]. Assume $\dim Z_{i-1} = n - i + 1$. By construction, for $i \in [0\,..\,m]$ every Z_i is a variety. By the check in line 05, we know that $Z'_i \neq Z_{i-1}$. We assume that g_i is irreducible; otherwise, the following argument applies to each of its irreducible component. Calling V the variety defined by the zeroes of the polynomial g_i, we know that $\dim V = n - 1$ [15, Exercise 2.8]. Observe that $Z'_i = Z_{i-1} \cap V$. If we decompose $Z'_i = V_1 \cup \ldots \cup V_l$, we can apply [15, Theorem 7.2] to argue that $\dim V_k \geq (n - i + 1) + (n - 1) - n = n - i$ for every k. If by contradiction $\dim V_k > n - i$, then $V_k \subseteq Z'_i \subsetneq Z_{i-1}$ contradicts the irreducibility of Z_{i-1}.

Finally, we show that, with probability at least d^{-n}, $\mathcal{B}$ perfectly simulates the n-SHS$_d[p]$ problem to $\mathcal{A}$ under the same secret $\boldsymbol{a}$ of the problem $\mathcal{B}$ is run against.

We argue that while $(1, \boldsymbol{a}) \in Z$ the simulation is correct. Line 05 returns 1 if $g(Z) = \{0\}$, which in particular means $g'(\boldsymbol{a}) = g(1, \boldsymbol{a}) = 0$. Line 07 returns 0 if $Z' = \emptyset$: In fact, if by contradiction $g(1, \boldsymbol{a}) = 0$, then $(1, \boldsymbol{a}) \in Z' = \emptyset$. The last return statement, in line 17, is just the bit returned by the real oracle Eval and then forwarded.

Consider the simulated oracle SimEval, and assume the condition in line 09 is triggered. Observe that if $(1, \boldsymbol{a}) \in Z$ at start, then there exists an index $h \in [1\,..\,l]$ such that $(1, \boldsymbol{a}) \in V_h$. By induction, this shows that there exists a sequence of indices $h_1, \ldots, h_m$ which, if picked in this order in line 14, make the simulation correct.

To conclude, we show that the probability the indices $h_1, \ldots, h_n$ are picked by the simulation is at least d^{-n}. Note that it is sufficient to prove this when conditioning the probability on the randomness of the adversary and of game n-SHS$_d[p]$. The only randomized procedure remaining is the choice of index j in line 14, whose randomness is freshly picked on each execution. The result then follows from the law of total probability.

We define the quantities used in the rest of the proof. Let j_i be the value assigned to variable j after a full execution of SimEval the i-th time the query to Eval is answered with 1, and $t_0 = 1$. For $i \in [0\,..\,m]$, we define events $C_i = \{j_k = h_k \text{ for all } 1 \leq k \leq i\}$. Since $n \geq m$, the proof is concluded if we show

$$\Pr[C_m] \geq d^{-m} \geq d^{-n} \enspace .$$

Assume now that event C_{i-1} occurs, which implies the actions of $\mathcal{B}$ are deterministic until the i-th time the query to Eval is answered with 1. We define t_i to be the (deterministic) degree of variety V_{h_i} and f_i to be the homogenization of the (deterministically chosen) polynomial queried by $\mathcal{A}$, that is, g. Then, by definition and by Eq. (2):

$$\Pr[C_i \mid C_{i-1}] \geq \frac{t_i}{t_{i-1} \deg f_i} \enspace . \tag{3}$$

This allows us to lower bound the winning probability by:

$$\Pr[C_m] = \prod_{i=1}^{m} \Pr[C_i \mid C_{i-1}] \geq \prod_{i=1}^{m} \frac{t_i}{t_{i-1} \deg f_i} \geq \frac{t_m}{t_0 d^m} \geq d^{-m} \enspace .$$

To prove this inequality we used, in order: $C_i \subseteq C_{i+1}$ and $\Pr[C_0] = 1$, Eq. (3), simplification of telescopic product (no term is zero) and $\deg f_i \leq d$, and $t_m \geq 1$ as any degree and $t_0 = 1$. □

Finally, we use Lemma 3 to bound the advantage of any adversary against n-SHS$_d[p]$. Intuitively, this lemma observes that the output of the adversary is, barring randomness, a function of the answers to the oracle queries. The bound follows from counting all possible distinct input values to an adversary as a function of the oracle answers and the total possible values for the secret $\boldsymbol{a} \in \mathbb{Z}_p^n$. Lemma 3 is used to limit the number of possible input values to the adversary.

Statement and proof closely follow [23, Theorem 6].

Lemma 4. *Let n, d be any positive integers and p a prime number. Then for every adversary $\mathcal{A}$ against n-SHS$_d[p]$ that makes at most q queries to the group-operation oracle:*

$$\mathrm{Adv}_p^{n\text{-shs}_d}(\mathcal{A}) \leq \left(\frac{d}{p}\right)^n \sum_{i=0}^{n} \binom{q}{i} \ .$$

Proof. We apply Lemma 3 to build an adversary $\mathcal{B}$ from $\mathcal{A}$. $\mathcal{B}$ makes exactly q queries to Eval (if less, we can modify the adversary to make additional queries to the polynomial 1), of which at most n return 1. Moreover

$$\mathrm{Adv}_p^{n\text{-shs}_d}(\mathcal{A}) \leq d^n \mathrm{Adv}_p^{n\text{-shs}_d}(\mathcal{B}) \ .$$

In the rest of the proof we work exclusively with adversary $\mathcal{B}$ to show that

$$\mathrm{Adv}_p^{n\text{-shs}_d}(\mathcal{B}) \leq \frac{1}{p^n} \sum_{i=0}^{n} \binom{q}{i} \ , \tag{4}$$

which proves the original statement.

Note that it is sufficient to prove (4) for any deterministic adversary $\mathcal{B}$. The result for probabilistic adversaries follows from the law of total probability.

Since $\mathcal{B}$ is deterministic with fixed initial input, its output only depends on the answers to its hypersurface queries. Let $\boldsymbol{b} \in \{0,1\}^q$ be any bit vector. For convenience we define the function $\mathrm{out}(\boldsymbol{b})$ to return the output of (deterministic, fixed-input) adversary $\mathcal{B}$ when its i-th query to Eval is answered with $\boldsymbol{b}[i]$.

Assume next to run game $\mathrm{G}_p^{n\text{-shs}_d}(\mathcal{B})$ under a fixed point $\boldsymbol{a} \in \mathbb{Z}_p^n$. The queries of $\mathcal{B}$ to Eval, as well as their responses $\boldsymbol{b}$, are fixed. Thus, we define the function $B(\boldsymbol{a})$ to be the function returning the vector $\boldsymbol{b}$ as described.

Let $\boldsymbol{a'}$ be the random variable representing the sampling of the vector $\boldsymbol{a} \in \mathbb{Z}_p^n$ in $\mathrm{G}_p^{n\text{-shs}_d}(\mathcal{B})$. Then, the distribution of $\boldsymbol{a'}$ is uniform over a set of size p^n. We proceed to compute a bound on the winning probability of $\mathcal{B}$:

$$\begin{aligned}
\Pr[\mathrm{G}_p^{n\text{-shs}_d}(\mathcal{B})] &= \Pr[\mathrm{out}(B(\boldsymbol{a'})) = \boldsymbol{a'}] \\
&= \sum_{\boldsymbol{a} \in \mathbb{Z}_p^n} \Pr[\boldsymbol{a'} = \boldsymbol{a}] \Pr[\mathrm{out}(B(\boldsymbol{a'})) = \boldsymbol{a'} \mid \boldsymbol{a'} = \boldsymbol{a}] \\
&= \frac{1}{p^n} \sum_{\boldsymbol{a} \in \mathbb{Z}_p^n} \Pr[\mathrm{out}(B(\boldsymbol{a})) = \boldsymbol{a}] \ .
\end{aligned}$$

Observe that for any $\boldsymbol{a}$ there exists a $\boldsymbol{b}$ such that $B(\boldsymbol{a}) = \boldsymbol{b}$. We can split the latter sum as follows:

$$\Pr[\mathrm{G}_p^{n\text{-shs}_d}(\mathcal{B})] \leq \frac{1}{p^n} \sum_{\boldsymbol{b}} \sum_{\boldsymbol{a}: B(\boldsymbol{a}) = \boldsymbol{b}} \Pr[\mathrm{out}(\boldsymbol{b}) = \boldsymbol{a}] \ . \tag{5}$$

Note that

$$\sum_{\boldsymbol{a}: B(\boldsymbol{a}) = \boldsymbol{b}} \Pr[\mathrm{out}(\boldsymbol{b}) = \boldsymbol{a}] \leq 1 \ . \tag{6}$$

Indeed, $\Pr[\mathrm{out}(\boldsymbol{b}) = \boldsymbol{a}]$ can be either 1 or 0, since function out is deterministic, and there exists at most one value of $\boldsymbol{a}$ which satisfies the expression $\mathrm{out}(\boldsymbol{b}) = \boldsymbol{a}$, namely $\mathrm{out}(\boldsymbol{b})$.

Combining (5) with (6) we obtain:

$$\Pr[\mathrm{G}_p^{n\text{-shs}_d}(\mathcal{B})] \leq \frac{1}{p^n} \sum_{\boldsymbol{b}} 1$$

To expand the previous expression we need to count all possible values for $\boldsymbol{b}$. Recall that at most n queries of $\mathcal{B}$ to Eval return 1. Counting all possible vectors of q bits of which at most n are 1 yields:

$$\Pr[\mathrm{G}_p^{n\text{-shs}_d}(\mathcal{B})] \leq \frac{1}{p^n} \sum_{i=0}^{n} \binom{q}{i} \qquad \square$$

We conclude by proving Theorem 3. The proof simply combines the bound in the previous lemma with a combinatorial result by Yun.

Proof (Theorem 3). [23, Theorem 7] states that

$$\sum_{i=1}^{n} \binom{q'}{i} \leq \frac{1}{2}\left(\frac{eq'}{n}\right)^n$$

for all positive integers q', n with $1 \leq n \leq q'$. Hence, by Lemma 4, every adversary $\mathcal{B}$ against n-SHS$_d[p]$ that makes at most q' hypersurface queries has advantage bounded by

$$\mathrm{Adv}_p^{n\text{-shs}_d}(\mathcal{B}) \leq \left(\frac{d}{p}\right)^n + \frac{1}{2}\left(\frac{edq'}{np}\right)^n \ .$$

We apply Lemma 2 to obtain our result, noting that the reduction makes less than $q' = q_{\mathrm{Eval}} + (n+q+1)^2/2$ queries to the hypersurface oracle. □

Remark 2. The bound on the advantage against (n,n)-d-PolyDL[$\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}$] of Theorem 3 extends to (m,n)-d-PolyDL[$\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}$], where $m \lneq n$. This is done by a simple tight reduction between problems (m,n)-d-PolyDL[$\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}$] and (m,m)-d-PolyDL[$\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}$]. The reduction for standard multi-instance discrete logarithm is in [22, Section 3]. To simulate oracle Eval, the reduction simply forwards the queries to its own oracle.

5.3 Generic Hardness of Medium-Granularity (m,n)-GapDL

We present an explicit bound on the concrete security of n-out-of-n gap discrete logarithm in the generic group model in the medium-granularity setting.

The bound is constructed by observing that we can simulate the medium-granularity game starting from the high-granularity one. Then, we apply the "high-granularity" Corollary 1 after counting the additional group queries by the simulation.

The following lemma states the existence of a reduction from (m,n)-GapDL in the medium-granularity setting to (m,n)-GapDL in the high-granularity setting.

Lemma 5. *Let* GGen *be a group-generating algorithm that generates groups of at least size p, and let m, n be two positive integers such that $m \leq n$. Then for every adversary $\mathcal{A}$ against (m,n)-*GapDL[$\mathsf{GGen}, \mathtt{med}$] *there exists an adversary $\mathcal{B}$ against (m,n)-*GapDL[$\mathsf{GGen}, \mathtt{high}$] *such that*

$$\mathrm{Adv}^{(m,n)\text{-gdl}}_{\mathsf{GGen},\mathtt{high}}(\mathcal{B}) \geq \mathrm{Adv}^{(m,n)\text{-gdl}}_{\mathsf{GGen},\mathtt{med}}(\mathcal{A}) \ .$$

Moreover, calling q_{DDH} the number of queries of $\mathcal{A}$ to DDH*, adversary $\mathcal{B}$ makes at most $2(2n+q_{\mathrm{DDH}})(\log p+1)$ group operations in addition to those made by $\mathcal{A}$, and the same amount of queries to* DDH*.*

Proof. Adversary $\mathcal{B}$ works as follows. $\mathcal{B}$ receives a group description $\mathcal{G} = (\mathbb{G}, p, g)$ and a n-size vectors $\boldsymbol{X} = g^{\boldsymbol{x}}$. It generates a uniform vector, $\boldsymbol{r} \leftarrow_{\$} (\mathbb{Z}_p \setminus \{0\})^n$, and computes $\boldsymbol{g'} \leftarrow g^{\boldsymbol{r}}$; $\boldsymbol{Z}[\cdot] \leftarrow \boldsymbol{X}[\cdot]^{\boldsymbol{r}[\cdot]}$. The computed elements correspond to the new group generators and the new challenges for $\mathcal{A}$. Queries to DDH on input of index i and group elements $\hat{X}, \hat{Y}, \hat{Z}$ are answered as follows. $\mathcal{B}$ queries its own oracle on input $\hat{X}, \hat{Y}^{1/\boldsymbol{r}[i]}, \hat{Z}$ and forwards the output to $\mathcal{A}$.

We prove that the oracle simulation is correct. Let $\hat{x}$, $\hat{y}$, $\hat{z}$ be such that $\hat{X} = g^{\hat{x}}$, $\hat{Y} = g^{\hat{y}}$, and $\hat{Z} = g^{\hat{z}}$ respectively. Since $\boldsymbol{g'}[i] = g^{\boldsymbol{r}[i]}$, $\mathcal{A}$ needs to receive output 1 if and only if

$$\left(g^{\boldsymbol{r}[i]}\right)^{\frac{\hat{x}}{\boldsymbol{r}[i]}\frac{\hat{y}}{\boldsymbol{r}[i]}} = \left(g^{\boldsymbol{r}[i]}\right)^{\frac{\hat{z}}{\boldsymbol{r}[i]}} \ ,$$

or, equivalently, $g^{\hat{x}\hat{y}/\boldsymbol{r}[i]} = g^{\hat{z}}$. This corresponds to a DDH query with respect to group generator g and input $\hat{X}, \hat{Y}^{1/\boldsymbol{r}[i]}, \hat{Z}$.

Next, we show that if $\mathcal{A}$ is successful then $\mathcal{B}$ is. If $\mathcal{A}$ is successful, then its output is a vector $\boldsymbol{x'}$ and a set $L = \{l_1, \ldots, l_m, \ldots\}$ such that $\boldsymbol{Z}[l_i] = (\boldsymbol{g'}[l_i])^{\boldsymbol{x'}[l_i]}$ for every i. If $\mathcal{A}$ is successful, $\boldsymbol{x'}$ is also the discrete logarithm vector of $\boldsymbol{X}$. In fact, for every i:

$$g^{\boldsymbol{x'}[l_i]} = \boldsymbol{g'}[l_i]^{\boldsymbol{x'}[l_i]/\boldsymbol{r}[l_i]} = \boldsymbol{Z}[l_i]^{1/\boldsymbol{r}[l_i]} = \boldsymbol{X}[l_i] \ .$$

$(L, \boldsymbol{x}')$ is the final output of $\mathcal{B}$, which wins the high-granularity game.

To conclude the proof, we count the additional group queries made by $\mathcal{B}$. The group-operation oracle is used when creating the new generators (n), challenges (n), and answering DDH queries (q_{DDH}). By using square-and-multiply we can upper bound the amount of queries by $2(2n + q_{\mathrm{DDH}})(\log p + 1)$. □

We use the previous reduction to compute the following bound.

Corollary 2. *Let* $\mathsf{GGen}_{\mathsf{gg}}$ *be a group-generating algorithm that generates generic groups of at least size* p*, and let* n *be a positive integer. Then for every generic adversary* $\mathcal{A}$ *against* (n, n)-GapDL$[\mathsf{GGen}_{\mathsf{gg}}, \mathtt{med}]$ *that makes at most* q *queries to the group-operation oracle and* q_{DDH} *queries to oracle* DDH*:*

$$\mathrm{Adv}^{(n,n)\text{-gdl}}_{\mathsf{GGen}_{\mathsf{gg}},\mathtt{med}}(\mathcal{A}) \leq \left(\frac{e(q + 4(q_{\mathrm{DDH}} + 2n)\log p)^2}{np}\right)^n \approx \left(\frac{q^2}{np}\right)^n \quad .$$

Proof. Combining Lemma 5, Corollary 1, and observing that, for $m = n$, the reduction makes additional $2(2n + q_{\mathrm{DDH}})(\log p + 1)$ group operations:

$$\begin{aligned}\mathrm{Adv}^{(n,n)\text{-gdl}}_{\mathsf{GGen}_{\mathsf{gg}},\mathtt{med}}(\mathcal{A}) &\leq \mathrm{Adv}^{(n,n)\text{-gdl}}_{\mathsf{GGen}_{\mathsf{gg}},\mathtt{high}}(\mathcal{B}) \\ &\leq \left(\frac{1}{p}\right)^n + \frac{1}{2}\left(\frac{e(q + 2(q_{\mathrm{DDH}} + 2n)(\log p + 1) + n + 1)^2 + 2eq_{\mathrm{DDH}}}{2np}\right)^n \quad .\end{aligned}$$

The previous expression can be simplified by noticing that, for the parameters we consider, $4n(\log p + 1) + n + 1 \leq 8n \log p$, $e(2q_{\mathrm{DDH}})^2 \geq 2eq_{\mathrm{DDH}}$, $2q_{\mathrm{DDH}}(\log p + 1) \leq 4q_{\mathrm{DDH}} \log p$, and $\sqrt[n]{2} \leq 2 \leq (8n \log p)^2/n$. □

Similarly to the previous concrete bounds, this result is optimal, namely there exists a generic adversary against (n, n)-GapDL$[\mathsf{GGen}_{\mathsf{gg}}, \mathtt{med}]$ needing $\Theta(\sqrt{np})$ group operations to achieve success probability 1. In fact, we can build an adversary against (n, n)-DL$[\mathsf{GGen}_{\mathsf{gg}}, \mathtt{med}]$ starting from an adversary against $(2n, 2n)$-DL$[\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}]$ by using it to retrieve (high-granularity) discrete logarithms of both challenges and generators in medium granularity, which can be used to retrieve the medium-granularity discrete logarithms.

The adversary works as follows: It receives the m group generators $\boldsymbol{g}$ and the n DL challenges $\boldsymbol{X}[\cdot] = \boldsymbol{g}[\cdot]^{\boldsymbol{x}[\cdot]}$. Then it runs the $(2n, 2n)$-DL$[\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}]$ adversary using as parameter the generator $g = \boldsymbol{g}[1]$ and, as DL challenges, the $2n$ group elements $\boldsymbol{g}, \boldsymbol{X}$. All queries to the gap oracle are forwarded prepending index 1. Assume the adversary correct, hence returning $\boldsymbol{r}, \boldsymbol{x}$ such that $\boldsymbol{g} = g^{\boldsymbol{r}}$ and $\boldsymbol{X} = g^{\boldsymbol{x}'}$. In particular $\boldsymbol{r}[\cdot]\boldsymbol{x}[\cdot] = \boldsymbol{x}'[\cdot]$, since

$$g^{\boldsymbol{r}[\cdot]\boldsymbol{x}[\cdot]} = \boldsymbol{g}[\cdot]^{\boldsymbol{x}[\cdot]} = \boldsymbol{X}[\cdot] = g^{\boldsymbol{x}'[\cdot]} \quad .$$

Retrieving $\boldsymbol{x}$ from the known $\boldsymbol{x}'$ and $\boldsymbol{r}$ concludes the reduction.

Corollary 4 is optimal: Using the adversary against $(2n, 2n)$-DL$[\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}]$ of Appendix F we see that for large p the best adversary against (n, n)-DL$[\mathsf{GGen}_{\mathsf{gg}}, \mathtt{med}]$ makes q queries, where $2\sqrt{2np} \leq q \leq \sqrt{np/e}$.

As in the high-granularity case, the technique of Ying and Kunihiro [22] can be used to extend the result to the case $m \neq n$.

5.4 Generic Hardness of Low-Granularity (m, n)-GapDL

In this section we present an explicit bound on the concrete security of m-out-of-n gap discrete logarithm in the generic group model in the low-granularity setting. Unlike for high and medium granularity, the CDH challenges belong to different groups: Since the n generic groups involved do not interact with each other, the bound can be derived by applying the single-instance bound n times. The bound is stated in the following theorem.

Theorem 4. *Let* $\mathsf{GGen}_{\mathsf{gg}}$ *be a group-generating algorithm that generates generic groups of at least size* p*, and let* m*,* n*,* q*,* q_{DDH} *and* q_i*,* $i \in [1\,..\,n]$*, be integers such that* $1 \leq m \leq n$*,* $q = q_1 + \ldots + q_n$*, and* q_i *is large (*$q_i \geq 2$ *and* $2q_i^2 \geq q_{\mathrm{DDH}} + 1$*). Then for every generic adversary* $\mathcal{A}$ *against the low-granularity* m*-out-of-*n *multi-instance computational Diffie-Hellman problem that makes at most* q_i *queries to the* i*-th group-operation oracle and* q_{DDH} *queries to the gap oracle:*

$$\mathrm{Adv}^{(m,n)\text{-gdl}}_{\mathsf{GGen}_{\mathsf{gg}},\mathtt{low}}(\mathcal{A}) \leq \left(\frac{4eq^2}{m^2 p}\right)^m \quad .$$

Proof. Let $\mathcal{A}$ be a generic adversary against (m,n)-GapDL[$\mathsf{GGen}_{\mathsf{gg}}, \mathtt{low}$], and let $\boldsymbol{x}$ be the random variable representing the (unique) solution vector with the discrete logarithms of the challenges. We model the n groups generated by $\mathsf{PGen}[\mathtt{low}]$ as independent generic groups of size $\boldsymbol{p}$, where $\boldsymbol{p}[i] \geq p$ for every $i \in [1\,..\,n]$. We can see the adversary as an adversary against the single-instance version in each of the n generic group. By Corollary 1, for $m = n = 1$ we know that:

$$\Pr\left[\mathcal{A} \text{ computes } \boldsymbol{x}[i]\right] \leq \frac{2}{p} + \frac{1}{2}\frac{e(q_i+2)^2 + 2eq_{\mathrm{DDH}}}{p} \leq 4e\frac{q_i^2}{p} \quad , \tag{7}$$

where the last inequality follows since q_i is large: $q_i \geq 2$ and $2q_i^2 \geq q_{\mathrm{DDH}} + 1$. Adversary $\mathcal{A}$ wins exclusively if it can identify at least m discrete logarithms. Let $L = \{l_1, \ldots, l_m, \ldots\}$ be the solution indices returned by $\mathcal{A}$. For the purpose of the proof, we define the set of indices $S = \{i_1, \ldots, i_m\}$ such that $q_{i_1} \geq \ldots \geq q_{i_m} \geq q_i$ for every $i \in [1\,..\,n] \setminus S$. We can bound the winning probability as:

$$\begin{aligned}
\Pr[\mathrm{G}^{(m,n)\text{-gcdh}}_{\mathsf{GGen}_{\mathsf{gg}},\mathtt{low}}(\mathcal{A})] &\leq \Pr\left[\mathcal{A} \text{ computes } \boldsymbol{x}[\{l_1, \ldots, l_m\}]\right] \\
&\leq \prod_{i=1}^{m} \Pr\left[\mathcal{A} \text{ computes } \boldsymbol{x}[l_i]\right] \\
&\leq \prod_{i \in S}\left(4\frac{q_i^2}{p}\right) \leq \left(\frac{4e}{p}\left(\frac{1}{m}\sum_{i\in S} q_i\right)^2\right)^m \leq \left(\frac{4eq^2}{m^2 p}\right)^m \quad .
\end{aligned}$$

To prove this bound we used, in order, the definition of $\mathrm{G}^{(m,n)\text{-gdl}}_{\mathsf{GGen}_{\mathsf{gg}},\mathtt{low}}$, the independence of each generic group, Eq. (7) and the fact that, after reordering, $q_{i_j} \leq q_{l_j}$ for every possible L and $j \in [1\,..\,m]$, and the inequality of arithmetic and geometric means, that is, for every nonnegative x_i such that $x_1 + \ldots + x_m = x$:

$$\prod_{i=1}^{m} x_i \leq \left(\frac{x}{m}\right)^m \quad . \qquad \square$$

Since the number of group operations performed by a (m,n)-GapDL adversary is typically large, we reckon the requirements $q_i \geq 2$ and $2q_i^2 \geq q_{\mathrm{DDH}} + 1$ are rather mild.

We argue that this result is optimal. In fact, the well-known baby-step giant-step algorithm solves $(1,1)$-DL with success probability 1 in $2\sqrt{p}$ group operations. We can build a generic adversary against game (m,n)-DL[$\mathsf{GGen}_{\mathsf{gg}}, \mathtt{low}$] by independently running the single-instance adversary on the first m instances. This adversary needs a total of $2m\sqrt{p}$ group operations to achieve success probability 1. Summing up, for large p the fastest generic adversary achieving overwhelming success probability in game (m,n)-GapDL[$\mathsf{GGen}_{\mathsf{gg}}, \mathtt{low}$] requires $m\sqrt{p/(8e)} \leq q \leq 2m\sqrt{p}$ group operations.

6 Generic Hardness of the Multi-Instance Gap Computational Diffie-Hellman Problem

In this section we derive lower bounds on the hardness of the m-out-of-n gap computational Diffie-Hellman problem in the generic group model for different granularities. We further argue that all derived bounds are optimal. Section 6.1 covers high, Section 6.2 medium, and Section 6.3 low granularity.

6.1 Generic Hardness of High-Granularity (m,n)-GapCDH

We work in the algebraic group model to show that the generic lower bound on the hardness of high-granularity (m,m)-GapDL carries over to high-granularity (m,n)-GapCDH. Concretely, Theorem 5 provides an algebraic reduction from (m,n)-GapCDH[$\mathsf{GGen}, \mathtt{high}$] to (m,m)-GapDL[$\mathsf{GGen}, \mathtt{high}$]. Then, an application of Corollary 1 establishes the desired bound on (m,n)-GapCDH.

In this section we work with high-granularity problems, in which the group description $\mathcal{G} = (\mathbb{G}, p, g)$ is shared by all instances. For ease of notation, we treat $\mathcal{G}$ as an implicit parameter of the system until the end of this section.

The algebraic reduction from (m,n)-GapCDH to (m,m)-GapDL in the high-granularity setting is sketched below.

Theorem 5. *Let* GGen *be a group-generating algorithm that generates groups of at least size* p*, and let* m*,* n *be two positive integers such that* $m \leq n \leq p$*. Then for every algebraic adversary* $\mathcal{A}$ *against* (m,n)-GapCDH$[\mathsf{GGen}, \mathtt{high}]$ *there exists an algebraic adversary* $\mathcal{B}$ *against* (m,m)-GapDL$[\mathsf{GGen}, \mathtt{high}]$ *such that*

$$\mathrm{Adv}^{(m,m)\text{-gdl}}_{\mathsf{GGen},\mathtt{high}}(\mathcal{B}) \geq 2^{-m}\mathrm{Adv}^{(m,n)\text{-gcdh}}_{\mathsf{GGen},\mathtt{high}}(\mathcal{A}) \ .$$

Moreover, $\mathcal{B}$ *makes at most* $2n(m+2)(\log p + 1)$ *group operations in addition to those made by* $\mathcal{A}$*, and the same amount of queries to* DDH*.*

Despite the seemingly sizeable reduction loss of 2^m, we argue that the factor is small in the context of the final security bounds. In fact, as seen in Section 5, the advantage in breaking (m,m)-GapDL decreases exponentially with m. This renders the exponential contribution of the factor 2^m irrelevant, as the following concrete bound on the hardness of (m,n)-GapCDH$[\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}]$ shows.

Corollary 3. *Let* $\mathsf{GGen}_{\mathsf{gg}}$ *be a group-generating algorithm that generates groups of at least size* p*, and let* m*,* n *be two positive integers such that* $m \leq n \leq p$*. Then for every generic adversary* $\mathcal{A}$ *against* (m,n)-GapCDH$[\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}]$ *that makes at most* q *queries to the group-operation oracle and* q_{DDH} *queries to the gap oracle:*

$$\mathrm{Adv}^{(m,n)\text{-gcdh}}_{\mathsf{GGen}_{\mathsf{gg}},\mathtt{high}}(\mathcal{A}) \leq \left(\frac{2e(q+12mn\log p)^2 + 4eq_{\mathrm{DDH}}}{mp}\right)^m \approx \left(\frac{q^2}{mp}\right)^m \ .$$

Proof. Combining Theorem 5 with Corollary 1 via Lemma 1 and observing that the reduction in Theorem 5 makes additional $2n(m+2)(\log p+1)$ group operations and the same amount of DDH queries we get:

$$\mathrm{Adv}^{(m,m)\text{-gcdh}}_{\mathsf{GGen}_{\mathsf{gg}},\mathtt{high}}(\mathcal{A}) \leq \left(\frac{4}{p}\right)^m + \frac{1}{2}\left(\frac{2e(q+2n(m+2)(\log p+1)+m+1)^2 + 4eq_{\mathrm{DDH}}}{mp}\right)^m \ .$$

The previous expression can be simplified by noticing that, for the parameters we consider, $2n(m+2)(\log p+1)+m+1 \leq 12mn\log p$ and $4\sqrt[m]{2} \leq 8 \leq (12mn\log p)^2/m$. □

Similarly to the bound for computing discrete logarithms, this result is optimal. Namely, problem (m,n)-GapCDH$[\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}]$ can be solved computing $2\sqrt{mp}$ group operations with success probability 1 by extending the generic adversary against high-granularity DL in Appendix F. Summing up, for large p the fastest generic adversary solving (m,n)-GapCDH$[\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}]$ with overwhelming success probability requires $\sqrt{mp/(2e)} \leq q \leq 2\sqrt{mp}$ group operations.

In the following paragraph we introduce some notation needed to prove Theorem 5.

Algebraic-Group Notation. Let Z be any group element returned by an algebraic adversary. We write $[Z]_{a_1,\ldots,a_k}$ to represent its algebraic representation in terms of the input group elements $X_1,\ldots,X_k$, that is, $Z = X_1^{a_1}\ldots X_k^{a_k}$. Similarly, when we consider n-size output vectors of group elements $\boldsymbol{Z}$ we write $[\boldsymbol{Z}]_{A_1,\ldots,A_h,\boldsymbol{b_1},\ldots,\boldsymbol{b_k}}$ to indicate that $\boldsymbol{Z} = \boldsymbol{Y_1}^{A_1}\ldots\boldsymbol{Y_h}^{A_h}X_1^{\ b_1}\ldots X_k^{\ b_k}$, where $\boldsymbol{Y_i}$ are s_i-size input group-element vectors, X_i are input group element, A_i is a $s_i \times n$ matrix, and $\boldsymbol{b_i}$ are n-size vectors. For clarity, if $\boldsymbol{z}$ (resp. $\boldsymbol{y_i}$) is the n-size (resp. s_i-size) vector of discrete logarithms of $\boldsymbol{Z}$ (resp. $\boldsymbol{Y_i}$) and x_i is the discrete logarithm of X_i, then $\boldsymbol{z} = A_1\boldsymbol{y_1} + \ldots + A_h\boldsymbol{y_h} + x_1\boldsymbol{b_1} + \ldots + x_k\boldsymbol{b_k}$.

Proof Idea of Theorem 5. This proof extends the following simple single-instance reduction $\mathcal{B}$, in turn built from two reductions $\mathcal{B}_\emptyset$ and $\mathcal{B}_{\{1\}}$. The reductions build upon a CDH adversary $\mathcal{A}$. Adversary $\mathcal{A}$ receives $X = g^x$ and $Y = g^y$, and is tasked with computing $W = g^{xy}$. In the algebraic group model, $\mathcal{A}$ must return a representation of the output as a combination of its input, i.e., some elements $a,b,c \in \mathbb{Z}_p$ such that $W = X^aY^bg^c$. Rewriting this expression in the exponents, we obtain that, if $\mathcal{A}$ wins,

$$xy = ax + by + c \ .$$

Given a DL challenge $Z = g^z$, reduction $\mathcal{B}_\emptyset$ embeds the challenge as $X = Z$ and generates $Y = g^y$ by picking a random y. Then, $\mathcal{B}_\emptyset$ can compute the DL as $z = x = (y-a)^{-1}(by+c)$. However, $y-a$ might not be invertible. In this case, adversary $\mathcal{B}_{\{1\}}$ would be successful: It embeds the challenge as $Y = Z$ and returns a, which is a correct solution if $y-a$ is not invertible. Reduction $\mathcal{B}$ picks one of the

two subsets $I \subseteq \{1\}$ at random and runs $\mathcal{B}_I$. If the CDH adversary is successful, then $\mathcal{B}$ has at least probability $1/2$ of succeeding.

Case $n = m > 1$ is approached as follows. Again the reduction $\mathcal{B}$ is composed of components $\mathcal{B}_I$, where $I \subseteq [1\,..\,n]$. The DL challenge $\boldsymbol{Z}[i]$ is distributed as either $\boldsymbol{X}[i]$ or $\boldsymbol{Y}[i]$ according to whether $i \in I$, and all remaining values are picked by the reduction. The CDH adversary—if successful—returns square matrices A, B and vector $\boldsymbol{c}$ such that $\mathrm{diag}(\boldsymbol{y})\boldsymbol{x} = A\boldsymbol{x} + B\boldsymbol{y} + \boldsymbol{c}$, where $\mathrm{diag}(\boldsymbol{y})$ is the diagonal matrix with the elements of $\boldsymbol{y}$ on the diagonal. Rearranging, we obtain

$$(\mathrm{diag}(\boldsymbol{y}) - A)\boldsymbol{x} = B\boldsymbol{y} + \boldsymbol{c} \ .$$

Our goal is to iteratively decrease the dimension of this matrix equation. If $n \notin I$ adversary $\mathcal{B}_I$ expresses $\boldsymbol{x}[n]$ in terms of $\boldsymbol{x}[1\,..\,n-1]$. On the other hand, if $n \in I$ then it computes $\boldsymbol{y}[n]$. Whether this computation is correct depends on whether I is the right choice for A, B, and $\boldsymbol{c}$. More explicitly, from the last row of the previous matrix equation we get the expression

$$\begin{aligned}\boldsymbol{x}[n](\boldsymbol{y}[n] - A_{nn}) = \quad & (A_{n1}, \ldots, A_{n(n-1)})\boldsymbol{x}[1\,..\,n-1] + \\ & + (B_{n1}, \ldots, B_{n(n-1)})\boldsymbol{y}[1\,..\,n-1] + B_{nn}\boldsymbol{y}[n] + \boldsymbol{c}[n] \ .\end{aligned}$$

If the number $\boldsymbol{y}[n] - A_{nn}$ is not invertible (case $n \in I$), then adversary $\mathcal{B}_I$ can set $\boldsymbol{y}[n] = A_{nn}$. In the other case (case $n \notin I$) the adversary can replace the expression for $\boldsymbol{x}[n]$ into the remaining $n-1$ rows of the matrix. In this case, $\boldsymbol{y}[n]$ is known, and calling $\boldsymbol{x'} = (\boldsymbol{x}[1], \ldots, \boldsymbol{x}[n-1])$, $\boldsymbol{y'} = (\boldsymbol{y}[1], \ldots, \boldsymbol{y}[n-1])$, we have recovered again a matrix equation of the form

$$\mathrm{diag}(\boldsymbol{y'})\boldsymbol{x'} = A'\boldsymbol{x'} + B'\boldsymbol{y'} + \boldsymbol{c'}$$

of decreased dimension $n-1$. Repeating this argument, we arrive at an equation of dimension 1. At this point all elements of $\boldsymbol{y}$ are known to $\mathcal{B}_I$, which is then able to recover the elements of $\boldsymbol{x}$.

Note that there always exists, for every possible A, B, and $\boldsymbol{c}$, a set I for which the above procedure is successful, i.e., a set I such that, for every $i \in [1\,..\,n]$, the expression $i \in I$ is satisfied exactly if $\boldsymbol{y}[i] = (A_{(i)})_{ii}$, where $A_{(i)}$ is the i-th update of matrix A. Since adversary $\mathcal{B}$ picks $I \subseteq [1\,..\,n]$ at random and runs $\mathcal{B}_I$, the reduction loses a factor of 2^n.

The case $n \neq m$ adds more complexity to the proof. The reduction first expands the m DL challenges $\hat{\boldsymbol{Z}}$ to a vector $\boldsymbol{Z} = \hat{\boldsymbol{Z}}^V$ (plus some rerandomization) of length n. Here V is a $n \times m$ matrix for which each $m \times m$ submatrix is invertible.[8] This has two important consequences: Firstly, we can express any element of $\boldsymbol{Z}$ as a combination of any other fixed m elements of $\boldsymbol{Z}$. Secondly, retrieving any m DLs of $\boldsymbol{Z}$ allows the reduction to compute the DLs of the original $\hat{\boldsymbol{Z}}$. This has, however, an unintended side effect: We can still obtain an equation of the form $\mathrm{diag}(\boldsymbol{y})\boldsymbol{x} = A\boldsymbol{x} + B\boldsymbol{y} + \boldsymbol{c}$, where all terms are of size m (this is the role, in the reduction code, of the function `reduceMatrices`), but now $A, B, \boldsymbol{c}$ depend on the distribution of the challenges to $\boldsymbol{X}$ and $\boldsymbol{Y}$, that is, on the set I. This means that the reduction cannot simply compute the element $\boldsymbol{y}[i]$ as A_{ii} at each step. It has to answer the question: "Assuming the reduction was not trying to compute $\boldsymbol{y}[m]$, what would be the value for $\boldsymbol{y}[m]$ which would make it unable to compute $\boldsymbol{x}[m]$?" (In the reduction code, the answer is yielded by the function `computeDlog`.)

In the proof, the gap oracle of $\mathcal{A}$ is simply simulated by forwarding all queries to DDH.

Proof (Theorem 5). Let $\mathcal{A}$ be an adversary against (m, n)-GapCDH[`GGen`, `high`].

We proceed as follows: We build adversaries $\mathcal{B}_I$ depending on some set $I \subseteq [1\,..\,n]$. The formal code is in Fig. 10. The m discrete logarithm challenges $\hat{\boldsymbol{Z}}$ are expanded to n challenges $\boldsymbol{Z}$ as $\boldsymbol{Z} = \hat{\boldsymbol{Z}}^V g^{\boldsymbol{r}}$ for a $n \times m$ Vandermonde matrix[9] V and a uniform vector $\boldsymbol{r} \in \mathbb{Z}_p^n$. The adversary produces the CDH challenges $\boldsymbol{X}$ and $\boldsymbol{Y}$ for $\mathcal{A}$ by distributing the expanded DL challenges $\boldsymbol{Z}$ either in $\boldsymbol{X}$ (for indices in I) or in $\boldsymbol{Y}$ (otherwise), and generating the remaining components uniformly. $\mathcal{A}$ returns a solution for m of those challenges, specified with the set L, along with their linear representation in term of the CDH challenges (algebraic adversary). Next it rewrites the linear representation to only depend on $\boldsymbol{x}[L], \boldsymbol{y}[L]$ instead of the full $\boldsymbol{x}, \boldsymbol{y}$ This is achieved by function `reduceMatrices`. Adversary $\mathcal{B}_I$ then computes the

[8] This expansion technique is originally from the work of Ying and Kunihiro [22].

[9] An $n \times m$ Vandermonde matrix is defined starting from a vector $(a_1, \ldots, a_n) \in \mathbb{Z}_p^n$ as the matrix $V = \{a_i^{j-1}\}_{i,j \in [1\,..\,n] \times [1\,..\,m]}$. If the a_i are pairwise distinct, then every $m \times m$ submatrix of V is invertible. Note that there are enough distinct elements to create V, since $p \geq n$.

discrete logarithms of each expanded DL challenges embedded in $\boldsymbol{Y}$ (function `computeDlog`), which are in turn used to recover the DLs of the entries of $\boldsymbol{X}$ indexed by elements of L. From this it extracts the solution to the original DL challenge $\hat{\boldsymbol{Z}}$. Any DDH query of $\mathcal{A}$ is simply forwarded to $\mathcal{B}_I$'s own oracle. Adversary $\mathcal{B}$ is defined by picking a random I and running $\mathcal{B}_I$.

Further, we define events $E_{\hat{I}}$ for $\hat{I} \subseteq [1\,..\,m]$ through the membership test in Fig. 9 (which, we remark, is not part of the reduction). Event $E_{\hat{I}}$ occurs if and only if the test returns 1. (The role of set $\hat{I}$ can be thought as picking which subset of the elements $\boldsymbol{y}[L]$ are computed, where L is the list of CDH indices returned by $\mathcal{A}$.) Later we show that, given $I \subseteq [1\,..\,n]$, if $E_{\{i:l_i \in I\}}$ occurs and $\mathcal{A}$ wins, then adversary $\mathcal{B}_I$ is successful in the multi-instance DL game. Since, as we will show, for all possible indices $\hat{I}$ the events $E_{\hat{I}}$ cover the whole probability space, we show that if $\mathcal{A}$ succeeds then adversary $\mathcal{B}$ succeeds with probability 2^{-m}.

Test membership of $E_{\hat{I}}(\boldsymbol{Y}, L, \hat{K}, A', B')$ $\quad$ // $\boldsymbol{Y} \in \mathbb{G}^n; L \subseteq [1..n]; \hat{K} \subseteq [1..n-m]; A', B' \in \mathbb{Z}_p^{m \times n}$

00 parse $\boldsymbol{Y}$ as $g^{\boldsymbol{y}}$ // membership test need not be efficient
01 parse L as $\{l_1, \ldots, l_m\}$ // drop elements if $|L| > m$, return 0 if $|L| < m$
02 $T \leftarrow [1\,..\,n] \setminus L$
03 parse T as $\{t_1, \ldots, t_{n-m}\}$
04 $I \leftarrow \{t_k : k \in \hat{K}\} \cup \{l_i : i \in \hat{I}\}$
05 $(A, B) \leftarrow$ `reduceMatrices`(I) // define $m \times m$ matrices A, B from A' and B'
06 for d from m down to 1:
07 parse A as $\begin{pmatrix} A_{d-1} & A_d^{\mathrm{r}} \\ A_d^{\mathrm{l}} & a_d \end{pmatrix}$ // single out index d. A_{d-1} is a square matrix of size $d-1$
08 if $l_d \in I$:
09 if $\boldsymbol{y}[l_d] = a_d$: return 0
10 $A \leftarrow A_{d-1} + (\boldsymbol{y}[l_d] - a_d)^{-1} A_d^{\mathrm{r}} A_d^{\mathrm{l}}$ // update matrix A. Dimension decreases by 1
11 else:
12 $\bar{a}_d \leftarrow$ `computeDlog`$(I \cup \{l_d\}, d)$
13 if $\boldsymbol{y}[l_d] \neq \bar{a}_d$: return 0
14 $A \leftarrow A_{d-1}$ // update matrix A. Dimension decreases by 1
15 return 1

Function `reduceMatrices`(I') // $I' \subseteq [1\,..\,n]$, outputs two square matrices of dimension m
16 $J' \leftarrow [1\,..\,n] \setminus I'$
17 $V \leftarrow n \times m$ Vandermonde matrix
18 parse V as $\begin{pmatrix} V_L \\ V_{T \cap I'} \\ V_{T \cap J'} \end{pmatrix}$ // group together rows with indices in L, $T \cap I'$, and $T \cap J'$
19 parse A' as $\begin{pmatrix} A'_L & A'_{T \cap I'} & A'_{T \cap J'} \end{pmatrix}$ // group together columns with...
20 parse B' as $\begin{pmatrix} B'_L & B'_{T \cap I'} & B'_{T \cap J'} \end{pmatrix}$ // ... indices in L, $T \cap I'$, and $T \cap J'$
21 parse $A'_{T \cap I'} V_{T \cap I'} V_L^{-1}$ as $\begin{pmatrix} M_{I'} & M_{J'} \end{pmatrix}$ // V_L is always invertible if $m \leq p$
22 parse $B'_{T \cap J'} V_{T \cap J'} V_L^{-1}$ as $\begin{pmatrix} N_{I'} & N_{J'} \end{pmatrix}$
23 $\bar{A} \leftarrow A'_L + \begin{pmatrix} M_{I'} & 0_{J'} \end{pmatrix} + \begin{pmatrix} N_{I'} & 0_{J'} \end{pmatrix}$ // make $M_{I'}$ and $N_{I'}$ into square...
24 $\bar{B} \leftarrow B'_L + \begin{pmatrix} 0_{I'} & M_{J'} \end{pmatrix} + \begin{pmatrix} 0_{I'} & N_{J'} \end{pmatrix}$ // ... matrices adding zero columns
25 return $(\bar{A}, \bar{B})$

Function `computeDlog`(I', k) // $I' \subseteq [1\,..\,n]$, $k \in [1\,..\,m]$, outputs element of $\mathbb{Z}_p$
26 $(\bar{A}, \bar{B}) \leftarrow$ `reduceMatrices`(I') // as in line 05
27 for d from m down to k: // similar to line 06, stops at k
28 parse $\bar{A}$ as $\begin{pmatrix} \bar{A}_{d-1} & \bar{A}_d^{\mathrm{r}} \\ \bar{A}_d^{\mathrm{l}} & \bar{a}_d \end{pmatrix}$ // as in line 07
29 if $l_d \in I'$: // as in line 08
30 $\bar{A} \leftarrow \bar{A}_{d-1} + (\boldsymbol{y}[l_d] - \bar{a}_d)^{-1} \bar{A}_d^{\mathrm{r}} \bar{A}_d^{\mathrm{l}}$ // as in line 10
31 else: // as in line 11
32 $\bar{A} \leftarrow \bar{A}_{d-1}$ // as in line 14
33 return $\bar{a}_k$

Fig. 9. Events $E_{\hat{I}}$ for the proof of Theorem 5 with respect to index set $\hat{I} \subseteq [1\,..\,m]$.

Adversary $\mathcal{B}_I^{\mathrm{DDH}}(\mathcal{G}, \hat{\boldsymbol{Z}})$ $\quad$ // $\mathcal{G}$ group representation; $\hat{\boldsymbol{Z}} \in \mathbb{G}^m$

00 $\boldsymbol{r} \leftarrow_{\$} \mathbb{Z}_p^n$

01 $V \leftarrow n \times m$ Vandermonde matrix

02 $\boldsymbol{Z} \leftarrow \hat{\boldsymbol{Z}}^V g^{\boldsymbol{r}}$ $\quad$ // from m to n challenges: $\boldsymbol{z} = V\hat{\boldsymbol{z}} + \boldsymbol{r}$

03 $J \leftarrow [1\,..\,n] \setminus I$

04 $\boldsymbol{x}[I] \leftarrow \perp$; $\boldsymbol{X}[I] \leftarrow \boldsymbol{Z}[I]$ $\quad$ // allocate challenges to either $\boldsymbol{X}$ or $\boldsymbol{Y}$

05 $\boldsymbol{y}[J] \leftarrow \perp$; $\boldsymbol{X}[J] \leftarrow \boldsymbol{Z}[J]$

06 $\boldsymbol{x}[J] \leftarrow_{\$} \mathbb{Z}_p^{|J|}$; $\boldsymbol{X}[J] \leftarrow g^{\boldsymbol{x}[J]}$ $\quad$ // uniformly pick remaining entries

07 $\boldsymbol{y}[I] \leftarrow_{\$} \mathbb{Z}_p^{|I|}$; $\boldsymbol{Y}[I] \leftarrow g^{\boldsymbol{y}[I]}$

08 $(L, [\boldsymbol{W}]_{A',B',\boldsymbol{c}'}) \leftarrow_{\$} \mathcal{A}^{\mathrm{DDH}}(\mathcal{G}, \boldsymbol{X}, \boldsymbol{Y})$ $\quad$ // algebraic adversary: $\boldsymbol{w} = A'\boldsymbol{x} + B'\boldsymbol{y} + \boldsymbol{c}'$

09 parse L as $\{l_1, \ldots, l_m\}$ $\quad$ // drop elements if $|L| > m$, abort if $|L| < m$

10 $T \leftarrow [1\,..\,n] \setminus L$

11 $(A, B) \leftarrow \texttt{reduceMatrices}(I)$ $\quad$ // define $m \times m$ matrices A, B from A' and B'

12 $\boldsymbol{c} \leftarrow \boldsymbol{c}' + A'_{T\cap J}\boldsymbol{x}[T \cap J] + B'_{T\cap I}\boldsymbol{y}[T \cap I] + A'_{T\cap I}V_{T\cap I}V_L^{-1}\boldsymbol{r}[L] + A'_{T\cap I}\boldsymbol{r}[T \cap J] +$
$\quad - B'_{T\cap J}V_{T\cap J}V_L^{-1}\boldsymbol{r}[L] + B'_{T\cap J}\boldsymbol{r}[T \cap J]$ $\quad$ // $\boldsymbol{c}$ has size m. $\boldsymbol{x}[J]$ and $\boldsymbol{y}[I]$ are known

13 for d from m down to 1: $\quad$ // A, B square matrices, $\boldsymbol{c}$ vector: all of dimension d

14 $\quad$ parse A as $\begin{pmatrix} A_{d-1} & A_d^{\mathrm{r}} \\ A_d^{\mathrm{l}} & a_d \end{pmatrix}$ $\quad$ // single out index d. A_{d-1} is a square matrix of size $d-1$

15 $\quad$ parse B as $\begin{pmatrix} B_{d-1} & B_d^{\mathrm{r}} \\ B_d^{\mathrm{l}} & b_d \end{pmatrix}$

16 $\quad$ parse $\boldsymbol{c}$ as $\begin{pmatrix} \boldsymbol{c_{d-1}} \\ \boldsymbol{c}[d] \end{pmatrix}$

17 $\quad$ if $l_d \in I$: $\quad$ // $\boldsymbol{x}[l_d]$ unknown, $\boldsymbol{y}[l_d]$ known

18 $\quad\quad A \leftarrow A_{d-1} + (\boldsymbol{y}[l_d] - a_d)^{-1}A_d^{\mathrm{r}}A_d^{\mathrm{l}}$ $\quad$ // update $A, B, \boldsymbol{c}$. Dimension decreases by 1

19 $\quad\quad B \leftarrow B_{d-1} + (\boldsymbol{y}[l_d] - a_d)^{-1}A_d^{\mathrm{r}}B_d^{\mathrm{l}}$

20 $\quad\quad \boldsymbol{c} \leftarrow \boldsymbol{c_{d-1}} + B_d^{\mathrm{r}}\boldsymbol{y}[l_d] + (\boldsymbol{y}[l_d] - a_d)^{-1}A_d^{\mathrm{r}}(\boldsymbol{c}[d] + b_d\boldsymbol{y}[l_d])$

21 $\quad$ else: $\quad$ // $\boldsymbol{x}[l_d]$ known, $\boldsymbol{y}[l_d]$ unknown

22 $\quad\quad \bar{a}_d \leftarrow \texttt{computeDlog}(I \cup \{l_d\}, d)$

23 $\quad\quad \boldsymbol{y}[l_d] \leftarrow \bar{a}_d$ $\quad$ // compute the value of $\boldsymbol{y}[l_d]$

24 $\quad\quad A \leftarrow A_{d-1}$ $\quad$ // update $A, B, \boldsymbol{c}$. Dimension decreases by 1

25 $\quad\quad B \leftarrow B_{d-1}$

26 $\quad\quad \boldsymbol{c} \leftarrow \boldsymbol{c_{d-1}} + A_d^{\mathrm{r}}\boldsymbol{x}[l_d] + B_d^{\mathrm{r}}\boldsymbol{y}[l_d]$

27 for d from 1 up to m: $\quad$ // all m components of $\boldsymbol{y}[L]$ are known or computed

28 $\quad$ if $l_d \in I$: $\quad$ // inductively: $\boldsymbol{x}[\{l_1\,..\,l_{d-1}\}]$ is known, use it to recover $\boldsymbol{x}[l_d]$

29 $\quad\quad \boldsymbol{x}[l_d] \leftarrow (\boldsymbol{y}[l_d] - a_d)^{-1}(A_d^{\mathrm{l}}\boldsymbol{x}[\{l_1\,..\,l_{d-1}\}] + B_d^{\mathrm{l}}\boldsymbol{y}[\{l_1\,..\,l_{d-1}\}] + b_d\boldsymbol{y}[l_d] + \boldsymbol{c}[d])$

30 $\boldsymbol{z}[I] \leftarrow \boldsymbol{x}[I]$; $\boldsymbol{z}[J] \leftarrow \boldsymbol{y}[J]$

31 parse V as $\begin{pmatrix} V_L \\ V_T \end{pmatrix}$ $\quad$ // group together rows with indices in L and T

32 $\hat{\boldsymbol{z}} \leftarrow V_L^{-1}(\boldsymbol{z}[L] - \boldsymbol{r}[L])$

33 return $\hat{\boldsymbol{z}}$

Fig. 10. Adversary $\mathcal{B}_I$ for the proof of Theorem 5 with respect to adversary $\mathcal{A}$ and index set $I \subseteq [1\,..\,n]$. Functions `reduceMatrices` and `computeDlog` are defined in Fig. 9. Implicitly, if $\mathcal{B}_I$ fails to invert an element at any point of the execution then it returns $\perp$. Empty matrices are ignored in sums and yield empty matrices in products. Queries to DDH by $\mathcal{A}$ are forwarded directly.

We use Proposition 1 and Proposition 2 to compute the winning probability of $\mathcal{B}$. The proof of both statements is postponed until the end of this proof.

Proposition 1. *Let $I \subseteq [1\,..\,n]$. Consider one execution of $\mathcal{B}_I$, and let $L = \{l_1, \ldots, l_m\}$ be the set defined by the output of the adversary $\mathcal{A}$ in such execution. Let $\hat{I} = \{i : l_i \in I\}$. If both $\mathcal{A}$ wins game $\mathrm{G}^{(m,n)\text{-gcdh}}_{\mathsf{GGen},\mathsf{high}}$ and $E_{\hat{I}}$ occurs then $\mathcal{B}_I$ wins game $\mathrm{G}^{(m,m)\text{-gdl}}_{\mathsf{GGen},\mathsf{high}}$.*

Proposition 2. *Let $\boldsymbol{Y} \in \mathbb{G}^n$, $L \subseteq [1\,..\,n]$, $\hat{K} \subseteq [1\,..\,n-m]$ and $A', B' \in \mathbb{Z}_p^{m\times n}$. Then there exists an $\hat{I} \subseteq [1\,..\,m]$ such that $E_{\hat{I}}(\boldsymbol{Y}, L, \hat{K}, A', B') = 1$.*

Let I be the random variable representing the choice of the index set in $[1\,..\,n]$ by $\mathcal{B}$. Since all input to $\mathcal{A}$ is independent of I (lines 04 to 07), the same holds for the internal variables $\boldsymbol{Y}$, $L = \{l_1, \ldots, l_m\}$, $T = [1\,..\,n] \setminus L = \{t_1, \ldots, t_{n-m}\}$, A', and B' defined in the execution of adversary $\mathcal{B}_J$. Hence we can write:

$$\begin{aligned}
\mathrm{Adv}^{(m,m)\text{-gdl}}_{\mathsf{GGen},\mathsf{high}}(\mathcal{B}) &\geq \sum_{J\subseteq[1\,..\,n]} \Pr[I = J \wedge \mathrm{G}^{(m,m)\text{-gdl}}_{\mathsf{GGen},\mathsf{high}}(\mathcal{B}_J)] \\
&\geq \sum_{J\subseteq[1\,..\,n]} 2^{-n} \Pr[\mathrm{G}^{(m,m)\text{-gdl}}_{\mathsf{GGen},\mathsf{high}}(\mathcal{B}_J)] \\
&\geq 2^{-n} \sum_{J\subseteq[1\,..\,n]} \Pr[E_{\{i:l_i\in J\}}(\boldsymbol{Y}, L, \{i : t_i \in J\}, A', B') \wedge \mathrm{G}^{(m,n)\text{-gcdh}}_{\mathsf{GGen},\mathsf{high}}(\mathcal{A})] \\
&\geq 2^{-n} \sum_{\substack{\hat{J}\subseteq[1\,..\,m] \\ \hat{K}\subseteq[1\,..\,n-m]}} \Pr[E_{\hat{J}}(\boldsymbol{Y}, L, \hat{K}, A', B') \wedge \mathrm{G}^{(m,n)\text{-gcdh}}_{\mathsf{GGen},\mathsf{high}}(\mathcal{A})] \\
&\geq 2^{-n} \sum_{\hat{K}\subseteq[1\,..\,n-m]} \Pr[\mathrm{G}^{(m,n)\text{-gcdh}}_{\mathsf{GGen},\mathsf{high}}(\mathcal{A})] = 2^{-m}\mathrm{Adv}^{(m,n)\text{-gcdh}}_{\mathsf{GGen},\mathsf{high}}(\mathcal{A}) \ .
\end{aligned}$$

In computing the previous inequality we employed, in order: the definition of $\mathcal{B}$, the independence of the uniform random variable I from event $\mathrm{G}^{(m,m)\text{-gdl}}_{\mathsf{GGen},\mathsf{high}}(\mathcal{B}_J)$, Proposition 1, a change of variables from J to $(\hat{J}, \hat{K})$, Proposition 2, and the fact that the set $[1\,..\,n-m]$ has 2^{n-m} subsets.

To conclude the proof we count the number of operations carried out by adversary $\mathcal{B}$. The operation count of $\mathcal{B}$ coincides with that of $\mathcal{B}_I$ for the chosen I. Clearly, all operations of $\mathcal{A}$ are executed by $\mathcal{B}_I$. Moreover, the reduction generates n new group elements by uniformly selecting exponents in $\mathbb{Z}_p$ (lines 06 and 07). Using the square-and-multiply algorithm, this correspond to an upper bound of $2n(\log p + 1)$ group operations. Similarly, to expand the DL challenges in line 02 $\mathcal{B}$ computes $\hat{\boldsymbol{Z}}^V g^{\boldsymbol{r}}$. Computing $\hat{\boldsymbol{Z}}$ requires $2mn(\log p+1)$ group operations and computing $\boldsymbol{r}$ requires $2n(\log p+1)$ group operations. Overall, $\mathcal{B}$ needs at most $2n(m+2)(\log p + 1)$ group operations. □

Proof (Proposition 1). If adversary $\mathcal{A}$ wins, then

$$\mathrm{diag}(\boldsymbol{x}[L])\boldsymbol{y}[L] = A'\boldsymbol{x} + B'\boldsymbol{y} + \boldsymbol{c}' \ , \tag{8}$$

where $\mathrm{diag}(\boldsymbol{x}[L])$ denotes the diagonal matrix with the elements of $\boldsymbol{x}[L]$ in the diagonal. Lines 11 and 12 of Fig. 10 define the triple $(A, B, \boldsymbol{c})$. We show that

$$\mathrm{diag}(\boldsymbol{x}[L])\boldsymbol{y}[L] = A\boldsymbol{x}[L] + B\boldsymbol{y}[L] + \boldsymbol{c} \ . \tag{9}$$

Let $\boldsymbol{z}$ (resp. $\hat{\boldsymbol{z}}$) be such that $\boldsymbol{Z} = g^{\boldsymbol{z}}$ (resp. $\hat{\boldsymbol{Z}} = g^{\hat{\boldsymbol{z}}}$), where $\boldsymbol{Z}$ is defined in line 02 of $\mathcal{B}_I$. By definition $\boldsymbol{z} = V\hat{\boldsymbol{z}} + \boldsymbol{r}$. We split this equation componentwise, depending if the component index belongs to any of the (disjoint) sets L, $T \cap I$, or $T \cap J$, as in done in line 18 of Fig. 9. Here $J = [1\,..\,n] \setminus I$.

$$\begin{aligned}
\boldsymbol{z}[L] &= V_L\hat{\boldsymbol{z}} + \boldsymbol{r}[L] \\
\boldsymbol{z}[T \cap I] &= V_{T\cap I}\hat{\boldsymbol{z}} + \boldsymbol{r}[T \cap I] \\
\boldsymbol{z}[T \cap J] &= V_{T\cap J}\hat{\boldsymbol{z}} + \boldsymbol{r}[T \cap J] \ .
\end{aligned} \tag{10}$$

V is a Vandermonde matrix, hence any $m \times m$ submatrix is invertible. In particular, the submatrix V_L containing the m rows indexed by L is invertible. Let $\boldsymbol{x}, \boldsymbol{y}$ be such that $\boldsymbol{X} = g^{\boldsymbol{x}}$ and $\boldsymbol{Y} = g^{\boldsymbol{y}}$. Recovering $\hat{\boldsymbol{z}}$

from Eq. (10) and recalling how $\boldsymbol{z}$ is distributed between $\boldsymbol{x}$ and $\boldsymbol{y}$ (lines 04 and 05) we can rewrite the last two equations as

$$\begin{aligned}\boldsymbol{x}[T\cap I] &= V_{T\cap I}V_L^{-1}\boldsymbol{z}[L] - V_{T\cap I}V_L^{-1}\boldsymbol{r}[L] + \boldsymbol{r}[T\cap I]\\ \boldsymbol{y}[T\cap J] &= V_{T\cap J}V_L^{-1}\boldsymbol{z}[L] - V_{T\cap J}V_L^{-1}\boldsymbol{r}[L] + \boldsymbol{r}[T\cap J]\ .\end{aligned} \tag{11}$$

Next, we parse A' and B' as in lines 19 and 20 of Fig. 9 and we rewrite Eq. (8):

$$\begin{aligned}\operatorname{diag}(\boldsymbol{x}[L])\boldsymbol{y}[L] = \ & A'_L\boldsymbol{x}[L] + A'_{T\cap I}\boldsymbol{x}[T\cap I] + A'_{T\cap J}\boldsymbol{x}[T\cap J] + \\ & + B'_L\boldsymbol{x}[L] + B'_{T\cap I}\boldsymbol{y}[T\cap I] + B'_{T\cap J}\boldsymbol{y}[T\cap J] + \boldsymbol{c'}\ .\end{aligned} \tag{12}$$

We expand the term $A'_{T\cap I}\boldsymbol{x}[T\cap I]$ using first Eq. (11), and then the parsing in line 21 of Fig. 9 and the repartition of $\boldsymbol{z}[L]$ among $\boldsymbol{x}[L\cap I]$ and $\boldsymbol{y}[L\cap J]$:

$$\begin{aligned}A'_{T\cap I}\boldsymbol{x}[T\cap I] &= A'_{T\cap I}V_{T\cap I}V_L^{-1}\boldsymbol{z}[L] - A'_{T\cap I}V_{T\cap I}V_L^{-1}\boldsymbol{r}[L] + A'_{T\cap I}\boldsymbol{r}[T\cap I]\\ &= \begin{pmatrix}M_I & 0_J\end{pmatrix}\boldsymbol{x}[L] + \begin{pmatrix}0_I & M_J\end{pmatrix}\boldsymbol{y}[L] - A'_{T\cap I}V_{T\cap I}V_L^{-1}\boldsymbol{r}[L] + A'_{T\cap I}\boldsymbol{r}[T\cap I]\ .\end{aligned} \tag{13}$$

Following a similar procedure we also obtain:

$$B'_{T\cap J}\boldsymbol{y}[T\cap J] = \begin{pmatrix}N_I & 0_J\end{pmatrix}\boldsymbol{x}[L] + \begin{pmatrix}0_I & N_J\end{pmatrix}\boldsymbol{y}[L] - B'_{T\cap J}V_{T\cap J}V_L^{-1}\boldsymbol{r}[L] + B'_{T\cap J}\boldsymbol{r}[T\cap J]\ . \tag{14}$$

Finally, we merge Eq. (13) and Eq. (14) into Eq. (12) and rearrange the terms:

$$\begin{aligned}\operatorname{diag}(\boldsymbol{x}[L])\boldsymbol{y}[L] = \ & (A'_L + \begin{pmatrix}M_I & 0_J\end{pmatrix} + \begin{pmatrix}N_I & 0_J\end{pmatrix})\boldsymbol{x}[L] + \\ & + (B'_L + \begin{pmatrix}0_I & M_J\end{pmatrix} + \begin{pmatrix}0_I & N_J\end{pmatrix})\boldsymbol{y}[L] + \\ & + (A'_{T\cap J}\boldsymbol{x}[T\cap J] + B'_{T\cap I}\boldsymbol{y}[T\cap I] - A'_{T\cap I}V_{T\cap I}V_L^{-1}\boldsymbol{r}[L] + \\ & + A'_{T\cap I}\boldsymbol{r}[T\cap I] - B'_{T\cap J}V_{T\cap J}V_L^{-1}\boldsymbol{r}[L] + B'_{T\cap J}\boldsymbol{r}[T\cap J] + \boldsymbol{c'})\ .\end{aligned}$$

The bracketed expressions are exactly how A, B, and $\boldsymbol{c}$ are defined, see lines 23 and 24 of Fig. 9 and line 12 of Fig. 10. This proves Eq. (9).

Next, we prove that all the $\boldsymbol{y}[l_d]$ computed in line 23 of Fig. 10 are correct. This basically follows from how we defined $E_{\{i:l_i\in I\}}$ in the first place. We consider the loop in lines 13 to 26: Observe that the adversary never tries to invert a zero element. In fact, by definition of $E_{\{i:l_i\in I\}}$ we obtain that $\boldsymbol{y}[d]-a_d$ is nonzero for $l_d\in I$. (Similarly, the adversary does not abort when inverting $\boldsymbol{y}[l_d]-a_d$ in line 29.) On the other hand, by definition of $E_{\{i:l_i\in I\}}$ we get $\boldsymbol{y}[d]=\bar{a}_d$ for $d\notin I$, which is consistent with all computed DLs of $\boldsymbol{Y}$.

We show next that the computation of $\boldsymbol{x}$ in the loop of lines 27 to 29 of $\mathcal{B}_I$ remains consistent with $\boldsymbol{X}=g^{\boldsymbol{x}}$. Let $\boldsymbol{x_d}=\boldsymbol{x}[\{l_1,\ldots,l_d\}]$ and $\boldsymbol{y_d}=\boldsymbol{y}[\{l_1,\ldots,l_d\}]$ for any $d\in[1\,..\,m]$. We show by induction that at the start of the d-indexed iteration of the loop in lines 13 to 26 the following holds:

$$\operatorname{diag}(\boldsymbol{y_d})\boldsymbol{x_d} = A\boldsymbol{x_d} + B\boldsymbol{y_d} + \boldsymbol{c}\ . \tag{15}$$

The case $d=m$ is Eq. (9). Suppose that our statement holds for some $d\in[2\,..\,m]$. We rewrite Eq. (15): First by isolating the first $d-1$ equations from the last one; Then by parsing A, B, and $\boldsymbol{c}$ as in lines 14 to 16 of $\mathcal{B}_I$.

$$\operatorname{diag}(\boldsymbol{y_{d-1}})\boldsymbol{x_{d-1}} = A_{d-1}\boldsymbol{x_{d-1}} + A^{\mathrm{r}}_d\boldsymbol{x}[l_d] + B_{d-1}\boldsymbol{y_{d-1}} + B^{\mathrm{r}}_d\boldsymbol{y}[l_d] + \boldsymbol{c_{d-1}} \tag{16}$$

$$\boldsymbol{y}[l_d]\boldsymbol{x}[l_d] = A^{\mathrm{l}}_d\boldsymbol{x_{d-1}} + a_d\boldsymbol{x}[l_d] + B^{\mathrm{l}}_d\boldsymbol{y_{d-1}} + b_d\boldsymbol{y}[l_d] + \boldsymbol{c}[d]\ . \tag{17}$$

We consider now two cases, depending on whether l_d belongs to I. If so, we can rewrite Eq. (17) as

$$\boldsymbol{x}[l_d] = (\boldsymbol{y}[l_d]-a_d)^{-1}(A^{\mathrm{l}}_d\boldsymbol{x_{d-1}} + B^{\mathrm{l}}_d\boldsymbol{y_{d-1}} + b_d\boldsymbol{y}[l_d] + \boldsymbol{c}[d])\ . \tag{18}$$

By replacing $\boldsymbol{x}[l_d]$ in Eq. (16) as per Eq. (18) we get:

$$\begin{aligned}\operatorname{diag}(\boldsymbol{y_{d-1}})\boldsymbol{x_{d-1}} = \ & (A_{d-1} + (\boldsymbol{y}[l_d]-a_d)^{-1}A^{\mathrm{r}}_dA^{\mathrm{l}}_d)\boldsymbol{x_{d-1}} + \\ & + (B_{d-1} + (\boldsymbol{y}[l_d]-a_d)^{-1}A^{\mathrm{r}}_dB^{\mathrm{l}}_d)\boldsymbol{y_{d-1}} + \\ & + (\boldsymbol{c_{d-1}} + B^{\mathrm{r}}_d\boldsymbol{y}[l_d] + (\boldsymbol{y}[l_d]-a_d)^{-1}A^{\mathrm{r}}_d(\boldsymbol{c}[d] + b_d\boldsymbol{y}[l_d]))\ .\end{aligned}$$

The terms in brackets are exactly how A, B, and $\boldsymbol{c}$ are defined for the next iteration of the loop, indexed by $d-1$ (see lines 18 to 20 of $\mathcal{B}_I$). This proves Eq. (15) for $d-1$ in the case $d \in I$.

On the other hand, for the case $d \notin I$ we can simply rewrite Eq. (16) as:

$$\begin{aligned}\operatorname{diag}(\boldsymbol{y_{d-1}})\boldsymbol{x_{d-1}} = \quad & (A_{d-1})\boldsymbol{x_{d-1}} + \\ & + (B_{d-1})\boldsymbol{y_{d-1}} + \\ & + (\boldsymbol{c_{d-1}} + A_d^{\mathrm{r}}\boldsymbol{x}[l_d] + B_d^{\mathrm{r}}\boldsymbol{y}[l_d]) \ .\end{aligned}$$

Again, the terms in brackets are exactly how A, B, and $\boldsymbol{c}$ are defined for the next iteration of the loop, which is indexed by $d-1$ (see lines 24 to 26 of $\mathcal{B}_I$). This proves Eq. (15) for $d-1$ in the case $d \notin I$, and consequently our inductive step.

Note that the extraction of the values $\boldsymbol{x}[l_d]$ for $d \in I$ in line 29 of $\mathcal{B}_I$ is correct. This is in fact Eq. (18). Lastly, we observe that the computed $\hat{\boldsymbol{z}}$ in line 32 of $\mathcal{B}_I$ corresponds to the discrete logarithms of $\hat{\boldsymbol{Z}}$. This is Eq. (10). □

Proof (Proposition 2). We prove the proposition by constructively showing the existence of $\hat{I}$. To find $\hat{I}$ we proceed as follows.

1. Set $\hat{I} \leftarrow [1\,..\,m]$.
2. Run the membership test of Fig. 9 with respect to $\hat{I}$ on $(\boldsymbol{Y}, L, \hat{K}, A', B')$.
3. If the test returns 1, then $\hat{I}$ is the sought set.
4. Else, let d be the loop index (line 06) at which "return 0" is triggered: Set $\hat{I} \leftarrow \hat{I} \setminus \{d\}$.
5. Repeat steps 2 to 5.

We claim that this algorithm terminates. To this end, it is sufficient to prove that every time steps 2 to 5 repeat the index d appearing in step 4 strictly decreases. Since $d \geq 1$, this proves our claim.

Let d be any index triggering "return 0" as per step 4, and assume that, after repeating steps 2 to 5, d^* is any index again triggering "return 0". (If the membership test returns 1 there is nothing to prove.) We want to prove that $d^* < d$. Similarly, let $\hat{I}, \hat{I}^* \subseteq [1\,..\,m]$ be the sets with respect to which the membership test in step 2 is run, I and I^* the corresponding set as defined in line 04, $J = [1\,..\,n] \setminus I$, $J^* = [1\,..\,n] \setminus I^*$, A and A^* the corresponding matrices as the A defined in line 05, and for every $i \in [1\,..\,m]$ let a_i (resp. a_i^*) and $\bar{a}_i$ (resp. $\bar{a}_i^*$) be the elements defined in lines 07 and 12 by running the membership test with respect to $\hat{I}$ (resp. $\hat{I}^*$).

First we show that $d^* \leq d$. For this it is sufficient to show that $a_i = a_i^*$ and $\bar{a}_i = \bar{a}_i^*$ for all $i > d$: In this case, the conditions triggering a "return 0" for indices $i < d$ remain unchanged when using $\hat{I}^*$ instead of $\hat{I}$. Combined with the fact that since $I = I^* \cup \{l_d\}$ it follows that the conditions in line 08 do not change for all $i > d$, we obtain that $d^* \leq d$.

To show that $a_i = a_i^*$ we consider how A is updated by the loops of the membership test. This is shown by induction on $i \in [d\,..\,m]$ by proving that, at the i-th iteration of the loop, matrices A and A^* only differ in the d-th column. We start with the base case $i = m$. In this case, the matrices are defined by running `reduceMatrices`, specifically in line 23 of Fig. 10. Since the input $\hat{K}$ only defines elements not in L, we know that $T \cap I = T \cap I^*$ and $T \cap J = T \cap J^*$, and hence all matrix multiplications in `reduceMatrices` are the same. The only difference between case $\hat{I}$ and $\hat{I}^*$ is the column grouping done in lines 21 and 22 of Fig. 9.

$$A - A^* = \left(M_{\{d\}}\ 0_{[1\,..\,m]\setminus\{d\}} \right) + \left(N_{\{d\}}\ 0_{[1\,..\,m]\setminus\{d\}} \right) \ .$$

This establishes the claim for $i = m$.

Next we prove the inductive step, assuming the case $i > d$ verified. We need to consider two cases: If $l_i \notin I$ (equivalent to $l_i \notin I^*$ for $i \neq d$) then the matrix is updated in line 10 by simply dropping the i-th row and column, which means that the updated A and A^* still differ only in the d-th column. If $l_i \in I$ (equivalent to $l_i \in I^*$ for $i \neq d$) then the matrix is updated in line 10. Since $a_i = a_i'$, by the induction hypothesis, we see that $\boldsymbol{y}[l_i] - a_i = \boldsymbol{y}[l_i] - a_i^*$. Next, consider the terms $A_i^{\mathrm{r}} A_i^{\mathrm{l}}$ and $A_i^{*\mathrm{r}} A_i^{*\mathrm{l}}$. Since $i \neq d$, by the induction hypothesis $A_i^{\mathrm{r}} = A_i^{*\mathrm{r}}$. Similarly, the row vector A_i^{l} differs from $A_i^{*\mathrm{l}}$ only at the d-th entry. It follows that $A_i^{\mathrm{r}} A_i^{\mathrm{l}}$ differs from $A_i^{*\mathrm{r}} A_i^{*\mathrm{l}}$ only in the d-th column. Consequently, the updated A and A^* differs only in the d-th column. Repeating the same argument replacing $\hat{I}$ and $\hat{I}^*$ with $\hat{I} \cup \{l_i\}$ and $\hat{I} \cup \{l_i\}$ shows that $\bar{a}_i = \bar{a}_i^*$.

Finally, we prove that $d \neq d^*$. We reach our goal by showing that $a_d = \bar{a}_d^*$: Since $l_d \in I$ and, by definition of d, in the case $\hat{I}$ the membership test returns 0 at the d-th iteration of the loop, we know

that $\boldsymbol{y}[l_d] = a_d$. If $a_d = \bar{a}_d^*$, since $l_d \notin I^*$ then the loop does not stop at the d-th iteration of the loop: In fact the condition $\boldsymbol{y}[l_d] \neq \bar{a}_d^*$ in line 13 is not triggered since $\boldsymbol{y}[l_d] = a_d = \bar{a}_d^*$. It remains to prove that $a_d = \bar{a}_d^*$. Consider the definition of `computeDlog` in Fig. 9. As the code comments highlight, the function traces the behavior of the membership test main loop. The function is run under input $I \cup \{l_d\}$. Since its input coincides with I^* and the loop is the same, the elements a_d and $\bar{a}_d^*$ must coincide. □

Remark 3. Note that using Corollary 3 with $q_{\mathrm{DDH}} = 0$ yields a generic lower bound on the hardness of the "standard" multi-instance CDH problem.

Further, oracle DDH plays a modest role in the proof of Theorem 5. One could define a "polycheck CDH" problem in the same fashion as it is done for discrete logarithm in Section 5 (in short, (m,n)-d-PolyCDH). It is then immediate to extend Theorem 5 to show the equivalence of games (m,n)-d-PolyCDH[$\mathsf{GGen}, \mathtt{high}$] and (m,n)-d-PolyDL[$\mathsf{GGen}, \mathtt{high}$] in the algebraic group model with the same loss of 2^m. Hence, the advantage of any adversary against game (m,n)-d-PolyCDH[$\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}$] can be bounded as in Corollary 3 with an additional multiplicative factor of $(d/2)^m$.

6.2 Generic Hardness of Medium-Granularity (m,n)-GapCDH

We present an explicit bound on the concrete security of m-out-of-n gap computational Diffie-Hellman in the generic group model in the medium-granularity setting. The main result of this section is similar to that in Section 6.1. Analogously to Section 5.3, the bound follows from observing that we can simulate the medium-granularity game starting from the high-granularity one. Then, we can apply Corollary 3 after counting the additional group queries by the simulation. For more details, we refer to Appendix G.1.

Corollary 4. *Let* $\mathsf{GGen}_{\mathsf{gg}}$ *be a group-generating algorithm that generates generic groups of at least size* p*, and let* m*,* n *be two positive integers such that* $m \leq n \leq p$*. Then for every generic adversary* $\mathcal{A}$ *against* (m,n)-GapCDH[$\mathsf{GGen}_{\mathsf{gg}}, \mathtt{med}$] *that makes at most* q *queries to the group-operation oracle and* q_{DDH} *queries to oracle* DDH*:*

$$\mathrm{Adv}^{(m,n)\text{-gdl}}_{\mathsf{GGen}_{\mathsf{gg}},\mathtt{med}}(\mathcal{A}) \leq \left(\frac{2e(q + 6(q_{\mathrm{DDH}} + 5mn)\log p)^2}{mp}\right)^m \approx \left(\frac{q^2}{mp}\right)^m \quad .$$

Similarly to the previous concrete bounds, this result is optimal, namely there exists a generic adversary against (m,n)-GapCDH[$\mathsf{GGen}_{\mathsf{gg}}, \mathtt{med}$] which needs $2\sqrt{2mp}$ group operations and achieves success probability 1. In fact, we can build an adversary against (m,n)-GapCDH[$\mathsf{GGen}_{\mathsf{gg}}, \mathtt{med}$] starting from an adversary against $(2m,2m)$-DL[$\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}$] that requires about the same amount of oracle queries, analogous to the approach of Section 5.3. Summing up, we obtain that for large p the fastest generic adversary achieving overwhelming success probability in game (m,n)-GapCDH[$\mathsf{GGen}_{\mathsf{gg}}, \mathtt{med}$] requires $\sqrt{mp/(2e)} \leq q \leq 2\sqrt{2mp}$ group operations.

6.3 Generic Hardness of Low-Granularity (m,n)-GapCDH

In this section we present an explicit bound on the concrete security of m-out-of-n gap computational Diffie-Hellman in the generic group model in the low-granularity setting. The bound is stated in the following theorem and is computed directly. The proof follows that of Theorem 4 and can be found in Appendix G.2.

Theorem 6. *Let* $\mathsf{GGen}_{\mathsf{gg}}$ *be a group-generating algorithm that generates generic groups of at least size* p*, and let* m*,* n*,* q*,* q_{DDH} *and* q_i*,* $i \in [1\,..\,n]$*, be integers such that* $1 \leq m \leq n$*,* $q = q_1 + \ldots + q_n$*, and* q_i *is large (*$q_i \geq 60\log p$ *and* $4q_i^2 \geq q_{\mathrm{DDH}}$*). Then for every generic adversary* $\mathcal{A}$ *against the low-granularity* m*-out-of-*n *multi-instance computational Diffie-Hellman problem that makes at most* q_i *queries to the* i*-th group-operation oracle and* q_{DDH} *queries to the gap oracle:*

$$\mathrm{Adv}^{(m,n)\text{-gcdh}}_{\mathsf{GGen}_{\mathsf{gg}},\mathtt{low}}(\mathcal{A}) \leq \left(\frac{4eq^2}{m^2p}\right)^m \quad .$$

Since the number of group operations performed by a (m,n)-GapCDH adversary is typically large, we reckon the requirements $q_i \geq 60\log p$ and $4q_i^2 \geq q_{\mathrm{DDH}}$ are rather mild.

We argue that this result is optimal. In fact, the well-known baby-step giant-step algorithm solves $(1,1)$-DL with success probability 1 in $2\sqrt{p}$ group operations. We can build a generic adversary against (m,n)-GapCDH[$\mathsf{GGen}_{\mathsf{gg}}, \mathtt{low}$] by independently running the single-instance adversary on the first m instances. This adversary needs a total of $2m\sqrt{p}$ group operations to achieve success probability 1. Summing up, for large p the fastest generic adversary achieving overwhelming success probability in game (m,n)-GapCDH[$\mathsf{GGen}_{\mathsf{gg}}, \mathtt{low}$] requires $m\sqrt{p/(8e)} \leq q \leq 2m\sqrt{p}$ group operations.

Acknowledgments

The authors are grateful to Masayuki Abe, Razvan Barbulescu, Mihir Bellare, Dan Boneh, Nadia Heninger, Tanja Lange, Alexander May, Bertram Poettering, Maximilian Rath, Sven Schäge, Nicola Turchi, and Takashi Yamakawa for their helpful comments. All authors were supported in part by the ERC Project ERCC (FP7/615074). Benedikt Auerbach was supported in part by the NRW Research Training Group SecHuman. Eike Kiltz was supported in part by DFG SPP 1736 Big Data and the DFG Cluster of Excellence 2092 CASA.

References

1. Abdalla, M., Bellare, M., Rogaway, P.: The oracle Diffie-Hellman assumptions and an analysis of DHIES. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 143–158. Springer, Heidelberg, Germany, San Francisco, CA, USA (Apr 8–12, 2001)
2. Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, J.A., Heninger, N., Springall, D., Thomé, E., Valenta, L., VanderSloot, B., Wustrow, E., Zanella-Béguelin, S., Zimmermann, P.: Imperfect forward secrecy: How Diffie-Hellman fails in practice. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015. pp. 5–17. ACM Press, Denver, CO, USA (Oct 12–16, 2015)
3. Barbulescu, R.: Algorithms for discrete logarithm in finite fields. Ph.D. thesis, University of Lorraine, Nancy, France (2013)
4. Barbulescu, R., Pierrot, C.: The multiple number field sieve for medium- and high-characteristic finite fields. LMS Journal of Computation and Mathematics 17(A), 230–246 (2014)
5. Bellare, M., Boldyreva, A., Micali, S.: Public-key encryption in a multi-user setting: Security proofs and improvements. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 259–274. Springer, Heidelberg, Germany, Bruges, Belgium (May 14–18, 2000)
6. Bellare, M., Ristenpart, T., Tessaro, S.: Multi-instance security and its application to password-based cryptography. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 312–329. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2012)
7. Bellare, M., Rogaway, P.: Optimal asymmetric encryption. In: Santis, A.D. (ed.) EUROCRYPT'94. LNCS, vol. 950, pp. 92–111. Springer, Heidelberg, Germany, Perugia, Italy (May 9–12, 1995)
8. Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 409–426. Springer, Heidelberg, Germany, St. Petersburg, Russia (May 28 – Jun 1, 2006)
9. Bernstein, D.J., Lange, T.: Batch NFS. In: Joux, A., Youssef, A.M. (eds.) SAC 2014. LNCS, vol. 8781, pp. 38–58. Springer, Heidelberg, Germany, Montreal, QC, Canada (Aug 14–15, 2014)
10. Coretti, S., Dodis, Y., Guo, S.: Non-uniform bounds in the random-permutation, ideal-cipher, and generic-group models. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 693–721. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2018)
11. Corrigan-Gibbs, H., Kogan, D.: The discrete-logarithm problem with preprocessing. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 415–447. Springer, Heidelberg, Germany, Tel Aviv, Israel (Apr 29 – May 3, 2018)
12. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 33–62. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2018)
13. Giacon, F., Kiltz, E., Poettering, B.: Hybrid encryption in a multi-user setting, revisited. In: Abdalla, M., Dahab, R. (eds.) PKC 2018, Part I. LNCS, vol. 10769, pp. 159–189. Springer, Heidelberg, Germany, Rio de Janeiro, Brazil (Mar 25–29, 2018)
14. Guillevic, A., Morain, F.: Discrete logarithms. In: Mrabet, N.E., Joye, M. (eds.) Guide to pairing-based cryptography. CRC Press - Taylor and Francis Group (Dec 2016)
15. Hartshorne, R.: Algebraic geometry. Springer-Verlag, New York-Heidelberg (1977)
16. Heninger, N., Durumeric, Z., Wustrow, E., Halderman, J.A.: Mining your Ps and Qs: Detection of widespread weak keys in network devices. In: 21st USENIX Security Symposium (2012)

17. Hitchcock, Y., Montague, P., Carter, G., Dawson, E.: The efficiency of solving multiple discrete logarithm problems and the implications for the security of fixed elliptic curves. International Journal of Information Security 3(2), 86–98 (Nov 2004)
18. Kuhn, F., Struik, R.: Random walks revisited: Extensions of Pollard's rho algorithm for computing multiple discrete logarithms. In: Vaudenay, S., Youssef, A.M. (eds.) SAC 2001. LNCS, vol. 2259, pp. 212–229. Springer, Heidelberg, Germany, Toronto, Ontario, Canada (Aug 16–17, 2001)
19. Maurer, U.M.: Abstract models of computation in cryptography (invited paper). In: Smart, N.P. (ed.) 10th IMA International Conference on Cryptography and Coding. LNCS, vol. 3796, pp. 1–12. Springer, Heidelberg, Germany, Cirencester, UK (Dec 19–21, 2005)
20. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT'97. LNCS, vol. 1233, pp. 256–266. Springer, Heidelberg, Germany, Konstanz, Germany (May 11–15, 1997)
21. Siegel, A.: Median bounds and their application. In: Tarjan, R.E., Warnow, T. (eds.) 10th SODA. pp. 776–785. ACM-SIAM, Baltimore, Maryland, USA (Jan 17–19, 1999)
22. Ying, J.H.M., Kunihiro, N.: Bounds in various generalized settings of the discrete logarithm problem. In: Gollmann, D., Miyaji, A., Kikuchi, H. (eds.) ACNS 17. LNCS, vol. 10355, pp. 498–517. Springer, Heidelberg, Germany, Kanazawa, Japan (Jul 10–12, 2017)
23. Yun, A.: Generic hardness of the multiple discrete logarithm problem. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 817–836. Springer, Heidelberg, Germany, Sofia, Bulgaria (Apr 26–30, 2015)

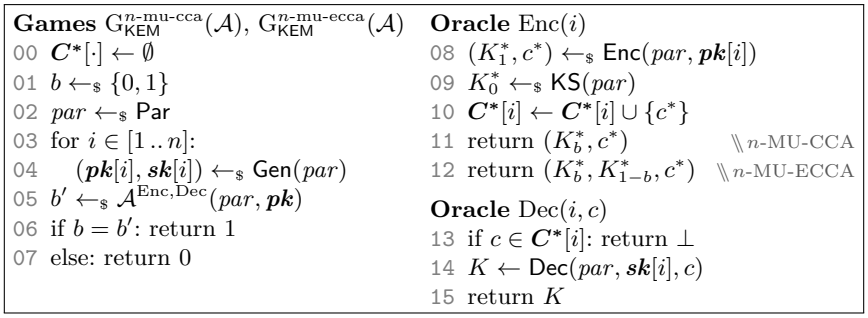

Fig. 11. Games $G^{n\text{-mu-cca}}_{\mathsf{KEM}}$ and $G^{n\text{-mu-ecca}}_{\mathsf{KEM}}$ modeling indistinguishability of encapsulated keys from random in the multi-user setting.

A Comparison Between Multi-User and Multi-Instance Security

In this section we compare the multi-user security notion introduced by Bellare et al. [5] to our 1-out-of-n multi-instance security, as defined in Section 3.1. Namely, we show that:

1. multi-user security tightly implies 1-out-of-n multi-instance security, and
2. 1-out-of-n multi-instance security implies multi-user security with a tightness loss of n.

The discussion that follows relates to CCA, however the same conclusions hold for CPA.

Both security notions aim to capture the hardness of breaking a cryptographic scheme in the presence of multiple users. The main difference between the two notions are the challenge bits: While 1-out-of-n security assigns to each public key its own challenge bit b_i, in the multi-user setting there exists a singe challenge bit b shared among all users. Another less determining difference is the output of the adversary: In the multi-user setting the adversary must simply output its guess of the shared bit b, while in our 1-out-of-n multi-instance setting the adversary outputs a set L containing the indices of the compromised users and the xor of their bits. If $L = \{i\}$, it means that the adversary returns the challenge bit for the i-th user. However, a successful adversary could return a set of size $|L| > 1$.

More formally, we present as game $G^{n\text{-mu-cca}}_{\mathsf{KEM}}(\mathcal{A})$ in Fig. 11 the CCA security definition for a key-encapsulation scheme KEM in the multi-user setting [13, Section 3.1]. We briefly describe the main differences between this security game and game $G^{(1,n)\text{-cca}}_{\mathsf{KEM}}$ from Fig. 1. Firstly, the game generates a single challenge bit in line 01 (cf. line 01, Fig. 1). Secondly, the output of the game, specified in lines 06 and 07, is 1 if and only if the adversary guesses the challenge bit correctly (cf. lines 07 to 09, Fig. 1). The advantage function is computed as in the multi-instance case with respect to game $G^{n\text{-mu-cca}}_{\mathsf{KEM}}$.

It is immediate to conclude that our KEM $(1, n)$-CCA implies multi-user security with a tightness loss of at most n. In fact, single-user security implies multi-user security with a tightness loss of n [13, Lemma 2] and $(1, n)$-CCA tightly implies single-user security, i.e., $(1, 1)$-CCA (Theorem 1, item 1).

To prove the other direction, we introduce a security notion, which we call "extended" multi-user CCA (n-MU-ECCA), that is equivalent to multi-user security. This notion is defined in Fig. 11 as $G^{n\text{-mu-ecca}}_{\mathsf{KEM}}$, and the advantage is computed as for standard multi-user security with respect to game $G^{n\text{-mu-ecca}}_{\mathsf{KEM}}$. The only difference between n-MU-CCA and n-MU-ECCA is that while in game $G^{n\text{-mu-cca}}_{\mathsf{KEM}}$ an adversary gets only a single key, either real or random, in game $G^{n\text{-mu-ecca}}_{\mathsf{KEM}}$ the adversary gets always both real and random keys, however the order of the two depends on the challenge bit.

The two notions are equivalent for every n. Clearly, n-MU-ECCA implies n-MU-CCA with no loss. We prove the other direction next.

Lemma 6. *Let n be a positive integer. Then for every adversary $\mathcal{A}$ against game $G^{n\text{-mu-ecca}}_{\mathsf{KEM}}$ there exists an adversary $\mathcal{B}$ against game $G^{n\text{-mu-cca}}_{\mathsf{KEM}}$ such that:*

$$\mathrm{Adv}^{n\text{-mu-cca}}_{\mathsf{KEM}}(\mathcal{B}) \geq \frac{1}{2}\mathrm{Adv}^{n\text{-mu-ecca}}_{\mathsf{KEM}}(\mathcal{A}) \ .$$

Moreover, $\mathcal{B}$ makes the same number of calls to its decryption oracle as $\mathcal{A}$ to its own.

Proof. The reduction works as follows. Adversary $\mathcal{B}$ forwards the received public keys and parameters to $\mathcal{A}$. Then, it generates a bit d. For any call to the encapsulation oracle of $\mathcal{A}$ on input i, adversary $\mathcal{B}$

calls Enc(i), obtaining (K_1, c). Then, it generates $K_0 \leftarrow_{\$} \mathsf{KS}(par)$ and returns (K_d, K_{1-d}, c) to $\mathcal{A}$. Queries to the decapsulation oracle are simply forwarded. Finally, let d' be the output of $\mathcal{A}$. Adversary $\mathcal{B}$ outputs 1 if and only if $d = d'$.

We compute the success probability of $\mathcal{A}$. Let b be the challenge bit implicitly used by game $\mathrm{G}^{n\text{-mu-cca}}_{\mathsf{KEM}}$. Then:

$$\begin{aligned}\mathrm{Adv}^{n\text{-mu-cca}}_{\mathsf{KEM}}(\mathcal{B}) &= 2\Pr[\mathrm{G}^{n\text{-mu-cca}}_{\mathsf{KEM}}(\mathcal{B})] - 1 \\ &= 2\Pr[\{b = 0 \wedge d \neq d'\} \cup \{b = 1 \wedge d = d'\}] - 1 \\ &= 2(\Pr[d \neq d' | d = 0]\Pr[d = 0] + \Pr[d = d' | d = 1]\Pr[d = 1]) - 1 \\ &= \Pr[d \neq d' | d = 0] + \Pr[d = d' | d = 1] - 1 \ .\end{aligned}$$

We compute the two probabilities above. Firstly, $\Pr[d \neq d' | d = 0] = 1/2$. In fact, if $d = 0$ then adversary $\mathcal{A}$ receives two random keys, which means that bit d is statistically hidden from $\mathcal{A}$. Secondly, $\Pr[d = d' | d = 1] = 1/2\mathrm{Adv}^{n\text{-mu-ecca}}_{\mathsf{KEM}}(\mathcal{A}) + 1/2$. In fact, if $d = 1$ then $\mathcal{B}$ always receives the real encapsulation key. Then, $\mathcal{B}$ correctly simulates game $\mathrm{Adv}^{n\text{-mu-ecca}}_{\mathsf{KEM}}$ to $\mathcal{A}$ under challenge bit d. The probability that $d' = d$, that is, the probability that $\mathcal{A}$ wins game $\mathrm{Adv}^{n\text{-mu-ecca}}_{\mathsf{KEM}}$, is $\Pr[\mathrm{G}^{n\text{-mu-ecca}}_{\mathsf{KEM}}] = 1/2\mathrm{Adv}^{n\text{-mu-ecca}}_{\mathsf{KEM}}(\mathcal{A}) + 1/2$.

Combining the previous observations we get:

$$\begin{aligned}\mathrm{Adv}^{n\text{-mu-cca}}_{\mathsf{KEM}}(\mathcal{B}) &= \Pr[d \neq d' | d = 0] + \Pr[d = d' | d = 1] - 1 \\ &= \frac{1}{2} + \left(\frac{1}{2}\mathrm{Adv}^{n\text{-mu-ecca}}_{\mathsf{KEM}}(\mathcal{A}) + \frac{1}{2}\right) - 1 = \frac{1}{2}\mathrm{Adv}^{n\text{-mu-ecca}}_{\mathsf{KEM}} \ .\end{aligned}$$ □

Finally, we show that extended multi-user security tightly implies 1-out-of-n multi-instance. This shows, in particular, that multi-user security tightly implies 1-out-of-n multi-instance.

Lemma 7. *Let n be a positive integer. Then for every adversary $\mathcal{A}$ against game $\mathrm{G}^{(1,n)\text{-cca}}_{\mathsf{KEM}}$ there exists an adversary $\mathcal{B}$ against game $\mathrm{G}^{n\text{-mu-ecca}}_{\mathsf{KEM}}$ such that:*

$$\mathrm{Adv}^{n\text{-mu-ecca}}_{\mathsf{KEM}}(\mathcal{B}) \geq \frac{1}{8}\mathrm{Adv}^{(1,n)\text{-cca}}_{\mathsf{KEM}}(\mathcal{A}) \ .$$

Moreover, $\mathcal{B}$ makes the same number of calls to its decryption oracle as $\mathcal{A}$ to its own.

Proof. The case $n = 1$ is immediate, since the two definitions are syntactically the same. Call s the random variable corresponding to the size of the set L output of $\mathcal{A}$ while running game $\mathrm{G}^{(1,n)\text{-cca}}_{\mathsf{KEM}}$.

We partition the probability space (randomness of $\mathrm{G}^{(1,n)\text{-cca}}_{\mathsf{KEM}}$) in the two events $E_1 = \{s = n\}$ and $E_2 = \{s \neq n\}$. We call $p_i = \Pr[E_i]$ for $i \in \{1, 2\}$. Then, we use these events to define the conditional advantage $a_i = 2\Pr[\mathrm{G}^{(1,n)\text{-cca}}_{\mathsf{KEM}}(\mathcal{A}) | E_i] - 1$ depending on the given adversary $\mathcal{A}$.

We can rewrite the advantage of $\mathcal{A}$ against game $\mathrm{G}^{(1,n)\text{-cca}}_{\mathsf{KEM}}$ as:

$$\mathrm{Adv}^{(1,n)\text{-cca}}_{\mathsf{KEM}}(\mathcal{A}) = 2\sum_{c=1}^{2}\Pr[\mathrm{G}^{(1,n)\text{-cca}}_{\mathsf{KEM}}(\mathcal{A}) | E_i]\Pr[E_i] - 1 = a_1p_1 + a_2p_2 \ .$$

We can conclude that there exists an index c such that $a_cp_c \geq 1/2\mathrm{Adv}^{(1,n)\text{-cca}}_{\mathsf{KEM}}(\mathcal{A})$. We present a different reduction for each possible value of c.

Case $c = 1$. We study the adversary $\mathcal{A}'$ that runs $\mathcal{A}$ and change its output as follows: If the (detectable) event E_1 occurs, then $\mathcal{A}'$ forwards the output of $\mathcal{A}$; otherwise, $\mathcal{A}$ returns $L = [1\,..\,n]$ and a random bit b'. Therefore, if event E_1 does not occur, then adversary $\mathcal{A}'$ breaks $(1, n)$-CCA (and also (n, n)-CCA) with probability $1/2$. Computing the advantage as an adversary against (n, n)-CCA we obtain:

$$\begin{aligned}\mathrm{Adv}^{(n,n)\text{-cca}}_{\mathsf{KEM}}(\mathcal{A}') &= 2\Pr[\mathrm{G}^{(n,n)\text{-cca}}_{\mathsf{KEM}}(\mathcal{A}')] - 1 \\ &= 2(\Pr[\mathrm{G}^{(1,n)\text{-cca}}_{\mathsf{KEM}}(\mathcal{A}') \wedge s = n] + \Pr[\mathrm{G}^{(1,n)\text{-cca}}_{\mathsf{KEM}}(\mathcal{A}') \wedge s \neq n]) - 1 \\ &= 2(\Pr[\mathrm{G}^{(1,n)\text{-cca}}_{\mathsf{KEM}}(\mathcal{A}) | E_1]\Pr[E_1] + 1/2(1 - \Pr[E_1])) - 1 \\ &= (2\Pr[\mathrm{G}^{(1,n)\text{-cca}}_{\mathsf{KEM}}(\mathcal{A}) | E_1] - 1)\Pr[E_1] = a_1p_1 \geq 1/2\mathrm{Adv}^{(1,n)\text{-cca}}_{\mathsf{KEM}}(\mathcal{A}) \ .\end{aligned}$$

Then we can use Theorem 1 to build from $\mathcal{A}'$ an adversary against $(1,1)$-CCA with the same advantage. The proof of case $c = 1$ follows, since any single-instance adversary can be trivially used as a multi-user adversary with the same advantage.

Case $c = 2$. Similar to the previous case, we build an adversary $\mathcal{A}'$ that outputs $L = \{1\}, b' \leftarrow_{\$} \{0,1\}$ when E_2 does not occur. With the same argument as before, we know that $\mathcal{A}'$ only outputs sets L such that $|L| \neq n > 1$, and $\mathcal{A}'$ has advantage:

$$\mathrm{Adv}^{(1,n)\text{-cca}}_{\mathsf{KEM}}(\mathcal{A}') \geq \frac{1}{2}\mathrm{Adv}^{(1,n)\text{-cca}}_{\mathsf{KEM}}(\mathcal{A}) \ .$$

We present our reduction next. Adversary $\mathcal{B}$ receives parameters par and public keys $\boldsymbol{pk}$ from game $\mathrm{G}^{n\text{-mu-ecca}}_{\mathsf{KEM}}$. $\mathcal{A}$ uniformly picks a set $S \subseteq [1\,..\,n]$ of size $\lceil n/2 \rceil$. Then, for every $i \in S$, $\mathcal{B}$ generates a key pair $(\boldsymbol{pk}[i], \boldsymbol{sk}[i])$, overwriting the challenge public key $\boldsymbol{pk}[i]$ previously received. Public keys $\boldsymbol{pk}$ are sent to $\mathcal{A}'$ along with the parameters par. Then, it generates bits $\boldsymbol{d} \leftarrow_{\$} \{0,1\}^n$ to simulate Enc queries to $\mathcal{A}'$, used as follows depending on input index $i \in [1\,..\,n]$: If $i \in S$, then $\mathcal{B}$ computes $(K_1, c) \leftarrow_{\$} \mathsf{Enc}(\boldsymbol{pk}[i])$, $K_0 \leftarrow_{\$} \mathsf{KS}(par)$ and returns $(K_{\boldsymbol{d}[i]}, c)$. Else, $\mathcal{B}$ calls its own oracle as $\mathrm{Enc}(i)$, obtaining keys K_0, K_1 and ciphertext c. The output of the simulated encapsulation oracle is then $(K_{\boldsymbol{d}[i]}, c)$. Decapsulation queries are handled with $\boldsymbol{sk}[i]$ when $i \in S$ or forwarded otherwise. Finally, given the output (L, b') of $\mathcal{A}'$, $\mathcal{B}$ returns the bit $b' \oplus \bigoplus_{i \in L} \boldsymbol{d}[i]$.

First, note that $\mathcal{A}'$ sees a correct simulation of game $\mathrm{G}^{(1,n)\text{-cca}}_{\mathsf{KEM}}$. In fact, calling b the challenge bit implicitly used in game $\mathrm{G}^{n\text{-mu-ecca}}_{\mathsf{KEM}}$, the reduction simulates user i under challenge bit $b_i = \boldsymbol{d}[i]$ if $i \in S$ or $b_i = b \oplus \boldsymbol{d}[i]$ if $i \notin S$, both of which are uniformly chosen.

Finally, if adversary $\mathcal{A}'$ wins against game $\mathrm{G}^{(1,n)\text{-cca}}_{\mathsf{KEM}}$, then adversary $\mathcal{B}$ wins against game $\mathrm{G}^{n\text{-mu-ecca}}_{\mathsf{KEM}}$ with probability at least $1/4$. In fact, the victory condition for $\mathcal{A}'$ is:

$$b' = \bigoplus_{i \in L} b_i = \bigoplus_{i \in L \cap S} \boldsymbol{d}[i] \oplus \bigoplus_{i \in L \setminus S} (b \oplus \boldsymbol{d}[i]) = \bigoplus_{i \in L \setminus S} b \oplus \bigoplus_{i \in L} \boldsymbol{d}[i] \ .$$

Note that if the event $E = \{|L \setminus S| \text{ is odd}\}$ occurs, then adversary $\mathcal{B}$ outputs the bit b, and otherwise the output bit is 0, independent of b. Event E is, moreover, hidden to $\mathcal{A}'$, since S is hidden to $\mathcal{A}'$. If we prove that $\Pr[E] \geq 1/4$, we conclude that, as in the theorem statement:

$$\mathrm{Adv}^{n\text{-mu-cca}}_{\mathsf{KEM}}(\mathcal{B}) \geq 1/4\mathrm{Adv}^{(1,n)\text{-cca}}_{\mathsf{KEM}}(\mathcal{A}') \geq 1/8\mathrm{Adv}^{(1,n)\text{-cca}}_{\mathsf{KEM}}(\mathcal{A}) \ .$$

To conclude, we show that $\Pr[E] \geq 1/4$. We rephrase the problem as follows: We start with $\lceil n/2 \rceil$ red balls (simulated keys) and $n - \lceil n/2 \rceil$ black balls (real keys), and we uniformly distribute them into a sequence of n balls. What is the probability that there are an odd number of black balls among the first $0 < k < n$ ($k = |L|$) balls of the sequence (event E)?

We split the probability space depending on the color of the first (f) and last (l) balls in the sequence. Assume both $f =$ black and $l =$ red, event which occurs with probability $\frac{\lceil n/2 \rceil}{n} \cdot \frac{n - \lceil n/2 \rceil}{n-1}$. Then there are $n-2$ balls left to distribute, $\lceil n/2 \rceil - 1$ red and $n - \lceil n/2 \rceil - 1$ black. Event E occurs if and only if there are an even number of black balls among the first $0 \leq k-1 \leq n-2$ redistributed balls: We call this probability p. Similarly, if $f =$ red and $l =$ black (occurring with probability $\frac{n - \lceil n/2 \rceil}{n} \cdot \frac{\lceil n/2 \rceil}{n-1}$) then E occurs if and only if there are an odd number of black balls among the first $k-1$ redistributed balls, that is, with probability $1-p$. Ignoring the remaining two cases for f and l we conclude:

$$\Pr[E] \geq \frac{\lceil n/2 \rceil}{n}\frac{n - \lceil n/2 \rceil}{n-1}p + \frac{n - \lceil n/2 \rceil}{n}\frac{\lceil n/2 \rceil}{n-1}(1-p) = \frac{\lceil n/2 \rceil}{n}\frac{n - \lceil n/2 \rceil}{n-1} \geq \frac{1}{4} \ ,$$

where the last inequality follows from the fact that $\lceil n/2 \rceil / n \geq 1/2$ and $(n - \lceil n/2 \rceil)/(n-1) \geq 1/2$ for all $n > 1$. □

B Omitted Proof of Section 3.2

In this section we describe the steps needed to prove Theorem 1.

We give an overview of the proof referencing to Fig. 2. We show in Lemma 8 how to build an adversary against (m, n') starting from an adversary against (m, n) with $n' < n$. In the diagram, this is needed to

move leftwards with respect to the dot representing the adversary against (m,n)-CPA, and is the source of the multiplicative loss ℓ. Lemma 9 shows how to move up slantwise along the diagonal of slope m/n. This is the source of the exponential loss k. We show the only tight security relations in the graph in Lemma 10. This is needed move to any point of the green section (advantage $\epsilon/2$ and ϵ). Finally, we prove Theorem 1 by combining the previous three lemmas to reach every point of Fig. 2.

The next lemma describes how to move leftwards from (m,n) in Fig. 2.

Lemma 8. *Let m, n, n' be positive integers such that $m \leq n' < n$, and let KEM be any KEM scheme. Then for every adversary $\mathcal{A}$ against game $\mathrm{G}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}$ there exists an adversary $\mathcal{B}$ against game $\mathrm{G}_{\mathsf{KEM}}^{(m,n')\text{-cpa}}$ with roughly the same running time as $\mathcal{A}$ such that*

$$\mathrm{Adv}_{\mathsf{KEM}}^{(m,n')\text{-cpa}}(\mathcal{B}) \geq \binom{n'}{m}\binom{n}{m}^{-1} \mathrm{Adv}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}(\mathcal{A}) \ .$$

An analogous statement holds between (m,n)-CCA and $(m,n-1)$-CCA, where adversary $\mathcal{B}$ queries the decryption oracle at most as often as $\mathcal{A}$.

Proof. Let $\mathcal{A}$ be an adversary against $\mathrm{G}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}$. To prove the statement it suffices to show that if $n > m$ there exists an adversary $\mathcal{B}'$ against game $\mathrm{G}_{\mathsf{KEM}}^{(m,n-1)\text{-cpa}}$ with roughly the same running time as $\mathcal{A}$ such that

$$\mathrm{Adv}_{\mathsf{KEM}}^{(m,n-1)\text{-cpa}}(\mathcal{B}') \geq \frac{n-m}{n} \cdot \mathrm{Adv}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}(\mathcal{A}) \ .$$

In fact, by applying this result $n - n'$ times we obtain:

$$\frac{n-m}{n} \cdots \frac{n-(n-n'-1)-m}{n-(n-n'-1)} = \frac{(n-m)!}{(n'-m)!}\frac{n'!}{n!} = \binom{n'}{m}\binom{n}{m}^{-1} \ .$$

In the rest of the proof we show the existence of $\mathcal{B}'$. Define $\epsilon = \Pr[\mathrm{G}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}(\mathcal{A})]$, $\epsilon_1 = \Pr[\mathrm{G}_{\mathsf{KEM}}^{(m,n)\text{-cpa}} \mid |L'| = m]$, and $\epsilon_2 = \Pr[\mathrm{G}_{\mathsf{KEM}}^{(m,n)\text{-cpa}} \mid |L'| \geq m+1]$. Moreover, call p be the probability that, while running game $\mathrm{G}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}$, adversary $\mathcal{A}$ outputs a set L' of size m Without loss of generality, we assume both that adversary $\mathcal{A}$ always outputs a set L' of size at least m and that $\epsilon_2 \geq 1/2$. In fact, any adversary against game $\mathrm{G}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}$ can be modified to satisfy both conditions without decreasing its advantage: If (L', b') is the output of $\mathcal{A}$, the two assumptions are achieved by outputting $([1\,..\,m], 0)$ every time that respectively $|L'| < m$ or $|L'| > m$. Under these restrictions, by the law of total probability we get:

$$\epsilon = p\epsilon_1 + (1-p)\epsilon_2 \ . \tag{19}$$

Adversary $\mathcal{B}'$ works as follows. It receives the keys $pk'_1, \ldots, pk'_{n-1}$ and parameters par as input and generates a key pair (pk'_n, sk'_n) and a uniform bit B. Then the order of the keys is rerandomized, obtaining keys $pk_1, \ldots, pk_n$. Adversary $\mathcal{A}$ is run on input $pk_1, \ldots, pk_n$ and par. If $\mathcal{A}$ calls the oracle Enc under the index corresponding to pk'_n, then $\mathcal{B}'$ computes c running $\mathsf{Enc}(pk'_n, m_B)$. Otherwise $\mathcal{B}'$ forwards the query to its oracle Enc for the corresponding index. Similarly, in the CCA case any decryption query for the simulated index is performed using sk'_n and for all other indices the query is forwarded to the oracle Dec. Eventually $\mathcal{A}$ outputs a set $\bar{L}'$ and a bit b'. Let L' be the set $\bar{L}'$ after reverting key index rerandomization. We define $L = L' \setminus \{n\}$. Moreover, depending if $n \in L'$, we define either $b = b' \oplus B$ or $b = b'$. The output of $\mathcal{B}'$ is (L, b).

First, observe that $\mathcal{A}$ sees a correct simulation of the game $\mathrm{G}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}$. We try to compute the success probability of $\mathcal{B}'$ by considering two distinct cases. Suppose that, while running $\mathcal{B}'$, we get $|L'| \geq m+1$. Then, by definition, $|L| \geq m$. It follows that:

$$\Pr[\mathrm{G}_{\mathsf{KEM}}^{(m,n-1)\text{-cpa}}(\mathcal{B}') \mid |L'| \geq m+1] = \Pr[\mathrm{G}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}(\mathcal{A}) \mid |L'| \geq m+1] = \epsilon_2 \ . \tag{20}$$

Suppose now that $|L'| = m$. Note that, since the key indices are rerandomized, the condition $n \notin L'$ is hidden from the adversary. Moreover, if $m \in L'$ then the bit output of $\mathcal{B}'$ is independent of the rest of the

game. We make use of this fact as follows:

$$\begin{aligned}\Pr[\mathrm{G}_{\mathsf{KEM}}^{(m,n-1)\text{-cpa}}(\mathcal{B}') \mid |L'| = m] &= \Pr[\mathrm{G}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}(\mathcal{A}) \wedge n \notin L' \mid |L'| = m] + \frac{1}{2}\Pr[n \in L' \mid |L'| = m] \\ &= \Pr[\mathrm{G}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}(\mathcal{A}) \mid |L'| = m] \cdot \Pr[n \notin L' \mid |L'| = m] + \frac{1}{2}\frac{m}{n} \\ &= \frac{n-m}{n}\epsilon_1 + \frac{m}{2n} \ . \end{aligned} \tag{21}$$

Combining, in order, the law of total probability, Eq. (21), Eq. (20), Eq. (19), and that $\epsilon_2 \geq 1/2$, we can write:

$$\begin{aligned}\mathrm{Adv}_{\mathsf{KEM}}^{(m,n-1)\text{-cpa}}(\mathcal{B}') &= 2\Pr[\mathrm{G}_{\mathsf{KEM}}^{(m,n-1)\text{-cpa}}(\mathcal{B}')] - 1 \\ &= 2\left(p\left(\frac{n-m}{n}\epsilon_1 + \frac{m}{2n}\right) + (1-p)\epsilon_2\right) - 1 \\ &= 2p\frac{n-m}{n}\epsilon_1 + p\frac{m}{n} + 2(1-p)\left(\frac{n-m}{n} + \frac{m}{n}\right)\epsilon_2 - \left(\frac{n-m}{n} + \frac{m}{n}\right) \\ &\geq \frac{n-m}{n}\big(2(p\cdot\epsilon_1 + (1-p)\cdot\epsilon_2) - 1\big) \\ &= \frac{n-m}{n}(2\epsilon - 1) = \frac{n-m}{n}\mathrm{Adv}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}(\mathcal{A}) \ . \end{aligned}$$

To conclude we study the CCA case. The same argument from CPA can be extended to CCA by simply either forwarding each encryption queries to the corresponding index or simulating the decryption oracle with the known secret keys. The final decryption query count for adversary $\mathcal{B}'$ is then at most the query count of each of the two executions of $\mathcal{A}$. The same holds for $\mathcal{B}$ □

The next lemma describes how to move up slantwise along the diagonal with slope m/n from (m, n) in Fig. 2.

Lemma 9. *Let m, n, k be positive integers such that $m \leq n$, and let KEM be any KEM scheme. Then for every adversary $\mathcal{A}$ against game $\mathrm{G}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}$ there exists an adversary $\mathcal{B}$ against game $\mathrm{G}_{\mathsf{KEM}}^{(km,kn)\text{-cpa}}$ with roughly k times the running time of $\mathcal{A}$ such that*

$$\mathrm{Adv}_{\mathsf{KEM}}^{(km,kn)\text{-cpa}}(\mathcal{B}) \geq \left(\mathrm{Adv}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}(\mathcal{A})\right)^k \ .$$

An analogous statement holds between (km, kn)-CCA and (m, n)-CCA, where additionally adversary $\mathcal{B}$ queries the decryption oracle at most k times more than $\mathcal{A}$.

Proof. Let $\mathcal{A}$ be an adversary against $\mathrm{G}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}$. Without loss of generality we assume that $\mathcal{A}$ always outputs a set L of size at least m. Moreover, we assume that $\Pr[\mathrm{G}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}(\mathcal{A}) \mid par] \geq 1/2$ for every possible parameter par. Otherwise, we can flip the output bit of the adversary for parameter par.

Adversary $\mathcal{B}$ works as follows: On input of a (km, kn)-CPA challenge $(par, \boldsymbol{pk})$ for $i \in [1\,..\,n]$, $\mathcal{B}$ sets $\boldsymbol{pk_j}[i] \leftarrow \boldsymbol{pk}[i+(j-1)n]$ for $j \in [1\,..\,k]$. (In this way, no vector $\boldsymbol{pk_j}$ shares an index with another vector $\boldsymbol{pk_{j'}}$ for $j \neq j'$.) Then it runs k independent copies $\mathcal{A}_j$ of adversary $\mathcal{A}$ on input $(par, \boldsymbol{pk_j})$, answering queries to Enc by using its own challenge oracle. Denote the output of $\mathcal{A}_j$ by (L_j, b'_j). Adversary $\mathcal{B}$ sets

$$b' \leftarrow \bigoplus_{j=1}^{k} b'_j \quad \text{and} \quad L \leftarrow \bigcup_{j=1}^{k} (L_j + jn) \ ,$$

where $L_j + jn$ is the set containing all elements of L shifted by the addend jn, and returns (L, b').

We now analyze $\mathcal{B}$'s advantage in winning game $\mathrm{G}_{\mathsf{KEM}}^{(km,kn)\text{-cpa}}$. Let $b = \bigoplus_{i\in L} \boldsymbol{b}[i]$, where $\boldsymbol{b}$ is the vector of challenge bits sampled in the game. $\mathcal{B}$ provides each $\mathcal{A}_j$ with a perfect simulation of the (m, n)-CPA game with respect to challenge bits $\boldsymbol{b_j}[i] = \boldsymbol{b}[i + jn]$. Since the indices used by each $\mathcal{A}_j$ are not shares by other copies, we know that

$$|L| = \sum_{j=1}^{k} |L_j| \geq km \ .$$

We argue next that, for the fixed parameters par:

$$2\Pr[b = b' \mid par] - 1 = (2\Pr[\mathrm{G}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{A}) \mid par] - 1)^k \ .$$

The proof follows by induction on k. In the case $k = 1$ the reduction simply forwards everything, and in particular $\Pr[b = b' \mid par] = \Pr[\mathrm{G}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{A}) \mid par]$. As inductive step, we prove the case $k + 1$. We implicitly split our reduction in two parts: The first computes the xor of the first k bits, and the second computes the $k + 1$-th bit. Then b is the xor of the output of the two parts. We introduce three new symbols: $b_{[1\,..\,k]} = \bigoplus_{i=1}^{k} b'_j$, $b_{[1\,..\,k]} = \bigoplus_{i \in L \cap [kn+1\,..\,(k+1)n]} \boldsymbol{b}[i]$, and $b_{k+1} = \bigoplus_{i \in L \cap [kn+1\,..\,(k+1)n]} \boldsymbol{b}[i]$. The probabilities that the bits computed by the two parts of the reduction are correct are then respectively $p_{[1\,..\,k]} = \Pr[b'_{[1\,..\,k]} = b_{[1\,..\,k]}]$ and $p_{k+1} = \Pr[b'_{k+1} = b_{k+1}] = \Pr[\mathrm{G}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{A}) \mid par]$. In particular, since by definition $b = b_{[1\,..\,k]} \oplus b_{k+1}$:

$$\begin{aligned}
\Pr[b = b' \mid par] &= \Pr[b'_{[1\,..\,k]} = b_{[1\,..\,k]}, b'_{k+1} = b_{k+1} \mid par] + \Pr[b'_{[1\,..\,k]} \neq b_{[1\,..\,k]}, b'_{k+1} \neq b_{k+1} \mid par] \\
&= p_{[1\,..\,k]} p_{k+1} + (1 - p_{[1\,..\,k]})(1 - p_{k+1}) \\
&= 2p_{[1\,..\,k]} p_{k+1} - p_{[1\,..\,k]} - p_{k+1} + 1 \ .
\end{aligned}$$

By our induction hypothesis we know that $2p_{[1\,..\,k]} - 1 = (2\Pr[\mathrm{G}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{A}) \mid par] - 1)^k$. Replacing values for $p_{[1\,..\,k]}$ and p_{k+1} in the following expression we get:

$$\begin{aligned}
2\Pr[b = b' \mid par] - 1 &= 4p_{[1\,..\,k]} p_{k+1} - 2p_{[1\,..\,k]} - 2p_{k+1} + 1 \\
&= (2p_{[1\,..\,k]} - 1)(2p_{k+1} - 1) \\
&= (2\Pr[\mathrm{G}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{A}) \mid par] - 1)^{k+1} \ ,
\end{aligned}$$

where we used the independence of the k instances $\mathcal{A}_j$ of $\mathcal{A}$, and the fact that $\mathrm{Adv}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{A}_j) = \mathrm{Adv}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{A})$. By using the previous formula and the law of total probability, we can write:

$$\begin{aligned}
\mathrm{Adv}^{(km,kn)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{B}') &= 2\Pr[b = b'] - 1 = 2 \sum_{par \in [\mathsf{Par}]} \Pr[b = b' \mid par] \Pr[par] - 1 \\
&= \sum_{par \in [\mathsf{Par}]} \left(2\Pr[\mathrm{G}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{A}) \mid par] - 1\right)^k \Pr[par] \\
&\geq \Big(\sum_{par \in [\mathsf{Par}]} \left(2\Pr[\mathrm{G}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{A}) \mid par] - 1\right) \Pr[par] \Big)^k \\
&= \left(2\Big(\sum_{par \in [\mathsf{Par}]} \Pr[\mathrm{G}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{A}) \mid par] \Pr[par] \Big) - 1 \right)^k = \left(\mathrm{Adv}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{A}) \right)^k \ ,
\end{aligned}$$

where the inequality step is consequence of the convexity of the map $x \mapsto x^k$ for x in the positive real numbers (recall we assume $\Pr[\mathrm{G}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{A}) \mid par] \geq 1/2$). To conclude, we study the CCA case. The same argument from CPA can be extended to CCA by simply forwarding each encryption queries to the corresponding index. The number of decryption queries by adversary $\mathcal{B}'$ is the total amount of queries by each of the two executions of $\mathcal{A}$. The bound on decryption queries of $\mathcal{B}$ is then k times that of $\mathcal{A}$. □

The next lemma describes how to move to any point in the green section (advantage $\epsilon/2$ and ϵ) from (m, n) in Fig. 2.

Lemma 10. *Let m, n, m', n' be positive integers such that $m' < m \leq n$, $m' \leq n'$ and $m'n \leq mn'$, and let KEM be any KEM scheme. Then for every adversary $\mathcal{A}$ against game $\mathrm{G}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}$ there exists an adversary $\mathcal{B}$ against game $\mathrm{G}^{(m',n')\text{-cpa}}_{\mathsf{KEM}}$ with roughly the same running time as $\mathcal{A}$ such that*

$$\mathrm{Adv}^{(m',n')\text{-cpa}}_{\mathsf{KEM}}(\mathcal{B}) \geq \frac{1}{2} \mathrm{Adv}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{A}) \ .$$

Additionally, if we also know that $n' - m' \geq n - m$ then the reduction does not lose the factor $1/2$. An analogous statement holds between (m', n')-CCA and (m, n)-CCA, where adversary $\mathcal{B}$ queries the decryption oracle at most as often as $\mathcal{A}$.

Proof. Let $\mathcal{A}$ be an adversary against $\mathrm{G}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}$. If $n' \geq n$ the result is trivial: Adversary $\mathcal{B}$ ignores the additional $n' - n$ user and uses $\mathcal{A}$ to break m instances of the first n users. Since $m' < m$, it follows that $\mathrm{Adv}_{\mathsf{KEM}}^{(m',n')\text{-cpa}}(\mathcal{B}) = \mathrm{Adv}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}(\mathcal{A})$. If $n' - m' \geq n - m$ then the result is again trivial: adversary $\mathcal{B}$ simulates $n' - n$ users to $\mathcal{A}$. Then, if successful, $\mathcal{A}$ returns a list L such that $|L| \geq m$. L contains at least m' indices that were not simulated. In fact, there are at most $n' - n$ simulated users, and in the worst case there are $m - n' + n \geq m'$ non-simulated users remaining in L.

We prove the remaining case. Let $n' < n$. $\mathcal{B}$ works as follows. $\mathcal{B}$ receives keys $pk'_1, \ldots, pk'_{n'}$ and parameters par as input, then generates key pairs $(pk'_{n'+1}, sk'_{n'+1})$ to (pk'_n, sk'_n) and corresponding uniform bits $B_{n'+1}, \ldots, B_n$. The order of the keys is rerandomized, which yields keys $pk_1, \ldots, pk_{n'}$. Adversary $\mathcal{A}$ is run on input $pk_1, \ldots, pk_n$ and par. If $\mathcal{A}$ calls the oracle Enc under the index corresponding to a key with known key pair (pk, sk), then $\mathcal{B}$ computes c running $\mathsf{Enc}(pk, m_B)$, where B is the bit corresponding to the key. Otherwise $\mathcal{B}$ forwards the query to its oracle Enc for the corresponding index. Similarly, any decryption query for the simulated index is performed using sk and for all other indices the query is forwarded to the oracle Dec. Eventually $\mathcal{A}$ outputs a set $\bar{L}'$ and a bit b'. Let L' be the set $\bar{L}'$ after reverting key index rerandomization. We define $L = L' \setminus [n' + 1 \,..\, n]$. Moreover, we define

$$b = b' \oplus \bigoplus_{i \in L' \cap [n'+1 \,..\, n]} B_i \ .$$

The output of $\mathcal{B}$ is (L, b).

Observe that if $\mathcal{A}$ wins then $\mathcal{B}$ wins if and only if $|L| \geq m$. In the rest of the proof we leverage the fact that the rerandomization hides the position of forwarded indices and simulated ones.

To compute $\Pr[|L| \geq m']$ we consider the following equivalent problem: From a set containing n' red balls (forwarded users) and $n - n'$ black balls (simulated users) we sample m elements without replacement. We want to prove that, calling X the number of red balls among the sampled elements,

$$\Pr[X \geq m'] \geq \frac{1}{2} \ .$$

Since by construction $\Pr[|L| \geq m'] = \Pr[X \geq m']$, proving the previous statement concludes the proof.

The random variable X follows a hypergeometric distribution of parameters (n, m, n'). To show that $\Pr[X \geq m'] \geq \frac{1}{2}$ it is sufficient to show that m' is smaller than the median of this distribution. The latter follows from [21, Corollary 2.3], where $R = n'$, $B = n' - n$, the two weights are the same, $r + b = m$ (and therefore $r = n' \cdot m/n \geq m'$). □

Finally, we combine the previous results to prove Theorem 1.

Proof (Theorem 1). Part 1 is Lemma 10.

To prove part 2, first we apply part 1 to prove that there exists $\mathcal{B}'$ such that:

$$\mathrm{Adv}_{\mathsf{KEM}}^{(m',\lceil nm'/m \rceil)\text{-cpa}}(\mathcal{B}') \geq \frac{1}{2}\mathrm{Adv}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}(\mathcal{A}) \ .$$

Then, since $\lceil nm'/m \rceil > n'$, we can apply Lemma 8. We obtain an adversary $\mathcal{B}$ from $\mathcal{B}'$ such that:

$$\begin{aligned}\mathrm{Adv}_{\mathsf{KEM}}^{(m',n')\text{-cpa}}(\mathcal{B}) &\geq \binom{n'}{m'}\binom{\lceil nm'/m \rceil}{m'}^{-1}\mathrm{Adv}_{\mathsf{KEM}}^{(m',\lceil nm'/m \rceil)\text{-cpa}}(\mathcal{B}') \\ &\geq \frac{1}{2}\binom{n'}{m'}\binom{\lceil nm'/m \rceil}{m'}^{-1}\mathrm{Adv}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}(\mathcal{A}) \ .\end{aligned}$$

Part 3 is proven by applying first Lemma 9 with $k = \lceil m'/m \rceil$. If m divides m', the result follows directly. Otherwise, there exists $\mathcal{B}'$ such that

$$\mathrm{Adv}_{\mathsf{KEM}}^{(km,kn)\text{-cpa}}(\mathcal{B}') \geq \left(\mathrm{Adv}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}(\mathcal{A})\right)^k \ .$$

Then, since $km > m'$ and $m'(kn) \geq (km)n'$ we use part 1 to build $\mathcal{B}$ such that:

$$\mathrm{Adv}_{\mathsf{KEM}}^{(m',n')\text{-cpa}}(\mathcal{B}) \geq \frac{1}{2}\mathrm{Adv}_{\mathsf{KEM}}^{(km,kn)\text{-cpa}}(\mathcal{B}') \geq \frac{1}{2}\left(\mathrm{Adv}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}(\mathcal{A})\right)^k \ .$$

Finally, part 4 is obtained by again applying first Lemma 9 with $k = \lceil m'/m \rceil$. Namely, there exists $\mathcal{B}'$ such that

$$\mathrm{Adv}^{(km,kn)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{B}') \geq \left(\mathrm{Adv}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{A})\right)^k \ .$$

Then, since $km > m'$ and $m'(kn) < (km)n'$ we use part 2 to build $\mathcal{B}$ such that:

$$\begin{aligned}\mathrm{Adv}^{(m',n')\text{-cpa}}_{\mathsf{KEM}}(\mathcal{B}) &\geq \frac{1}{2}\binom{n'}{m'}\binom{\lceil (kn)m'/(km)\rceil}{m'}^{-1}\mathrm{Adv}^{(km,kn)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{B}') \\ &\geq \frac{1}{2}\binom{n'}{m'}\binom{\lceil nm'/m\rceil}{m'}^{-1}\left(\mathrm{Adv}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{A})\right)^k \ .\end{aligned}$$

The same proof applies to the CCA case and yields the number of calls to the decryption oracle indicated in the statement. □

C A Scheme with Scaling Factor Below One

In this section we build a (toy) public-key encryption scheme PKE and we argue that its CPA scaling factor is upper bounded by m/n. The same construction can be applied for key-encapsulation mechanisms, but we use PKE for clarity.

We start from a perfectly secure scheme $\mathsf{PKE}'' = (\mathsf{Enc}'', \mathsf{Dec}'')$. (We leave parameter generation and key generation implicit, since they are not going to change in the following constructions.) We fix a probability $p \in [0,1]$ and we build a scheme PKE' by defining:

$$\mathsf{Enc}'(pk, m) = \begin{cases} (0, \mathsf{Enc}''(pk, m)) & \text{with probability } p \\ (1, m) & \text{with probability } 1-p \end{cases} \ .$$

The corresponding decryption procedure is immediate. Then we introduce two hash functions, namely H and F, and a large integer k, which are used in the next step. Importantly, F is a hard-to-compute function, which means that the running time of any adversary is dominated by the number of times F is computed. k is large in the sense that the adversaries we consider are not able to compute function F more than k times.[10] Scheme PKE is defined by:

$$\begin{aligned}\mathsf{Enc}(pk, m) = \big(&\mathsf{Enc}'(pk, m) \oplus H(pk, F(1)), \\ &\mathsf{Enc}'(pk, m) \oplus H(pk, F(2)), \\ &\ldots, \\ &\mathsf{Enc}'(pk, m) \oplus H(pk, F(k))\big) \ .\end{aligned}$$

Decryption corresponds to running Dec' of the first component after xoring out $H(pk, F(1))$.

We want to prove that, for $m \neq n$:

$$\mathrm{SF}^{(m,n)\text{-cpa}}_{\mathsf{PKE}} = \frac{\mathrm{MinTime}^{(m,n)\text{-cpa}}_{\mathsf{PKE}}}{\mathrm{MinTime}^{(1,1)\text{-cpa}}_{\mathsf{PKE}}} \leq \frac{m}{n} \ . \tag{22}$$

To this end, we upper bound first the advantage of a single-instance adversary that makes a single encryption query. We model H and F as a random oracles. This is used to recover a lower bound on the time q (i.e., number of queries to F) needed to win with at least probability $1-\alpha$. Then we give an explicit adversary against m-out-of-n multi-instance which runs in time $q' = mq/n$ and has success probability at least $1-\alpha$. If both facts hold for overwhelming probability $1-\alpha$, then Eq. (22) is verified.

Let $\mathcal{A}$ be an adversary against single-instance CPA that makes at most q queries to F. Let $c = (c'_1 \oplus h_1, \ldots, c'_k \oplus h_k)$ be the ciphertext obtained by running $\mathcal{A}$ on the CPA game, where c'_i is a ciphertext of PKE' and h_i is in the range of H. Each entry c'_i is statistically hidden unless H has been queried on input $(pk, F(i))$. Ignoring the probability of guessing $F(i)$, this means that $\mathcal{A}$ knows at most q

[10] This would make encryption unfeasible, since it calls F exactly k times. We can assume each user knows a trapdoor to compute $H(pk, F(\cdot))$.

ciphertexts c'_i, and the rest is hidden. Since PKE'' is assumed to be a perfect encryption scheme, an adversary can recover the original message from c'_i only with probability $1-p$. The probability that $\mathcal{A}$ wins is then at most

$$\Pr[\mathrm{G}^{(1,1)\text{-cpa}}_{\mathsf{PKE}}(\mathcal{A})] \leq 1-(1-(1-p))^q = 1-p^q \ .$$

In particular, if an adversary has advantage $1-\alpha$, then it makes at most $q = \log\alpha/\log p$ queries.

Next, we construct an adversary $\mathcal{A}$ against (m,n)-CPA that makes $q' = \frac{m}{n}q = \frac{m\log\alpha}{n\log p}$ queries such that $\Pr[\mathrm{G}^{(m,n)\text{-cpa}}_{\mathsf{PKE}}(\mathcal{A})] \geq 1-\alpha$ for $1-\alpha$ overwhelming, that is, α close to 0. Adversary $\mathcal{A}$ simply computes $F(1),\ldots,F(q')$ and uses them to check whether encrypting with Enc' returned the message in all first q' components of the ciphertexts it receives.

Adversary $\mathcal{A}$ wins if, for at least m users, at least one of the first q' components were not encrypted with Enc. Explicitly:

$$\begin{aligned}
\Pr[\mathrm{G}^{(m,n)\text{-cpa}}_{\mathsf{PKE}}(\mathcal{A})] &\geq \sum_{i=m}^{n}\binom{n}{i}\left(1-p^{q'}\right)^i\left(1-(1-p^{q'})\right)^{(n-i)} \\
&= 1-\sum_{i=0}^{m-1}\binom{n}{i}\left(1-p^{q'}\right)^i p^{q'(n-i)} \\
&= 1-\sum_{i=0}^{m-1}\binom{n}{i}(1-\alpha^{m/n})^i\alpha^{(n-i)m/n}
\end{aligned}$$

We use the previous inequality to show that $\Pr[\mathrm{G}^{(m,n)\text{-cpa}}_{\mathsf{PKE}}(\mathcal{A})] \geq 1-\alpha$:

$$\begin{aligned}
1-\alpha &\leq 1-\sum_{i=0}^{m-1}\binom{n}{i}(1-\alpha^{m/n})^i\alpha^{(n-i)m/n} \\
\alpha &\geq \sum_{i=0}^{m-1}\binom{n}{i}(1-\alpha^{m/n})^i\alpha^{(n-i)m/n} \qquad (23) \\
1 &\geq f(\alpha) \coloneqq \sum_{i=0}^{m-1}\binom{n}{i}(1-\alpha^{m/n})^i\alpha^{(n-i)m/n-1}
\end{aligned}$$

We argue that $f(0)=0$. In this case, since the function f is, by construction, continuous, Eq. (23) is true for α close to 0. To prove $f(0)=0$, we argue that $(n-i)m/n-1>0$. The rest follows by directly replacing values in the definition of f. Note that, since $i \leq m-1$:

$$(n-i)m/n-1 \geq (n-m+1)m/n-1$$

Moreover, since $n \neq m$, if $m \neq 1$:

$$(n-m+1)\frac{m}{n}-1 = \frac{(m-1)(n-m)}{n} > 0 \ .$$

If $m=1$, then Eq. (23) becomes $\alpha \geq \alpha$.

D Omitted Proof of Section 4

Proof (Theorem 2). Since the proofs for the three granularity variants differ only in the way adversary $\mathcal{B}$ sets up the parameters and public keys to be used by $\mathcal{A}$, we only give the full proof for the KEM variant $\mathsf{HEG}[\mathsf{GGen},\mathtt{high}]$. At the end of the proof we sketch how $\mathcal{B}$ has to be modified for the cases of medium and low granularity.

For simplicity, in this proof we assume that adversary $\mathcal{A}$ for every $i \in [1\,..\,n]$ poses at most one query to oracle $\mathrm{Enc}(i)$. However, using the random self-reducibility of the computational Diffie-Hellman problem the proof can be easily adapted to adversaries posing several queries of the form $\mathrm{Enc}(i)$.

Consider adversary $\mathcal{A}$ playing game $\mathrm{G}^{(m,n)\text{-cca}}_{\mathsf{HEG}[\mathsf{GGen},\mathtt{high}]}$ of Fig. 1. For $i \in [1\,..\,n]$ we denote by $\boldsymbol{c}^*[i]$ the ciphertext generated in line 10 as part of the challenge, which the adversary receives as response to its query $\mathrm{Enc}(i)$. By definition of $\mathsf{HEG}[\mathsf{GGen},\mathtt{high}]$, the ciphertext $\boldsymbol{c}^*[i]$ is an element of the group specified by the parameters, and its decapsulation with respect to secret key $\boldsymbol{sk}[i] = \boldsymbol{x}[i]$ is $\mathsf{Dec}(par, \boldsymbol{sk}[i], \boldsymbol{c}^*[i]) = \mathrm{RO}(\boldsymbol{pk}[i], \boldsymbol{c}^*[i], \boldsymbol{c}^*[i]^{\boldsymbol{x}[i]})$, where RO denotes the random oracle modeling the hash function H. Let Q denote the event

$$Q = \left\{ \mathcal{A} \text{ queries for } \mathrm{RO}(\boldsymbol{pk}[i], \boldsymbol{c}^*[i], \boldsymbol{c}^*[i]^{\boldsymbol{x}[i]}) \text{ for all } i \in L \right\} \ ,$$

where L is the list of indices output by the adversary at the end of the game. Note that, since hash function H is modeled as a random oracle, $\mathsf{Dec}(par, \boldsymbol{sk}[i], \boldsymbol{c}^*[i])$ is distributed uniformly on $\mathsf{KS}(par)$. Hence oracle Enc on input i returns a pair (K^*, c^*) consisting of the ciphertext c^* and an element K^* of $\mathsf{KS}(par)$, which is uniformly distributed both in the case that the corresponding challenge bit $\boldsymbol{b}[i]$ equals 0 or that it equals 1. So $\mathcal{A}$ only receives input which depends on $\boldsymbol{b}[i]$ if it queries both $\mathrm{Enc}(i)$ and $\mathrm{RO}(\boldsymbol{pk}[i], \boldsymbol{c}^*[i], \boldsymbol{c}^*[i]^{\boldsymbol{x}[i]})$.

Assume that event Q does not occur. In this case there exists an index $j \in L$ such that $\mathcal{A}$ does not query for $\mathrm{RO}(\boldsymbol{pk}[j], \boldsymbol{c}^*[j], \boldsymbol{c}^*[j]^{\boldsymbol{x}[j]})$. This implies that all inputs to $\mathcal{A}$—and hence also $\mathcal{A}$'s output b'—are independent of challenge bit $\boldsymbol{b}[j]$. Thus, when conditioning on $\neg Q$, the bit b' is independent of $\bigoplus_{i \in L} \boldsymbol{b}[i]$, which implies $\Pr[\mathrm{G}^{(m,n)\text{-cca}}_{\mathsf{HEG}[\mathsf{GGen},\mathtt{high}]}(\mathcal{A}) \mid \neg Q] = 1/2$. We obtain

$$\begin{aligned}
\mathrm{Adv}^{(m,n)\text{-cca}}_{\mathsf{HEG}[\mathsf{GGen},\mathtt{high}]}(\mathcal{A}) &= 2\Pr[\mathrm{G}^{(m,n)\text{-cca}}_{\mathsf{HEG}[\mathsf{GGen},\mathtt{high}]}(\mathcal{A})] - 1 \\
&= 2\Pr[\mathrm{G}^{(m,n)\text{-cca}}_{\mathsf{HEG}[\mathsf{GGen},\mathtt{high}]}(\mathcal{A}) \mid Q]\Pr[Q] + 2\Pr[\mathrm{G}^{(m,n)\text{-cca}}_{\mathsf{HEG}[\mathsf{GGen},\mathtt{high}]}(\mathcal{A}) \mid \neg Q]\Pr[\neg Q] - 1 \\
&\leq 2\Pr[Q] + (1 - \Pr[Q]) - 1 = \Pr[Q] \ . \qquad (24)
\end{aligned}$$

We conclude the proof by constructing an adversary $\mathcal{B}$ against the m-out-of-n multi-instance gap computational Diffie-Hellman game $\mathrm{G}^{(m,n)\text{-gcdh}}_{\mathsf{GGen},\mathtt{high}}$ of Fig. 4 in high granularity that provides $\mathcal{A}$ with a perfect simulation of game $\mathrm{G}^{(m,n)\text{-cca}}_{\mathsf{HEG}[\mathsf{GGen},\mathtt{high}]}$ and successfully computes a solution to its (m,n)-GapCDH challenge if event Q occurs. The description of $\mathcal{B}$ is in Fig. 12. Adversary $\mathcal{B}$ on input of a (m,n)-GapCDH challenge $(\boldsymbol{\mathcal{G}}, \boldsymbol{X}, \boldsymbol{Y})$ sets $par \leftarrow \boldsymbol{\mathcal{G}}[1]$ and uses $\boldsymbol{pk}[i] = \boldsymbol{X}[i]$ as the i-th public key. Then it runs $\mathcal{A}$ on input $(par, \boldsymbol{pk})$. Note that since we consider (m,n)-GapCDH in high granularity all group elements $\boldsymbol{X}[i]$ belong to the same group $\boldsymbol{\mathcal{G}}[1] = \boldsymbol{\mathcal{G}}[i]$. Hence $\boldsymbol{pk}$ has the correct distribution.

Adversary $\mathcal{B}$ provides $\mathcal{A}$ with a perfect simulation of the oracles Enc, Dec, RO by keeping track of three tables: T_{Dec} stores decapsulations corresponding to ciphertexts, T_{RO} stores hash values, and T_{DDH} stores Diffie-Hellman tuples, which it is able to identify using oracle DDH. The challenge ciphertext with respect to index i is set to $\boldsymbol{c}^*[i] = \boldsymbol{Y}[i]$. Hence if $\mathcal{A}$ queries for $\mathrm{RO}(\boldsymbol{pk}[i], \boldsymbol{c}^*[i], \boldsymbol{c}^*[i]^{\boldsymbol{x}[i]})$, then the last two elements of the input to RO are the i-th partial solution $\boldsymbol{g}[i]^{\boldsymbol{x}[i]\boldsymbol{y}[i]}$ to its (m,n)-GapCDH challenge. $\mathcal{B}$ identifies these tuples and stores them in a vector $\boldsymbol{Z'}$. When $\mathcal{A}$ at the end of game $\mathrm{G}^{(m,n)\text{-cca}}_{\mathsf{HEG}[\mathsf{GGen},\mathtt{high}]}$ outputs a list L and a bit b', adversary $\mathcal{B}'$ returns L and $\boldsymbol{Z'}$. If Q occurs then $\boldsymbol{Z'}$ contains all partial solutions to its (m,n)-GapCDH challenge corresponding to the indices in L. By Eq. (24) we obtain

$$\mathrm{Adv}^{(m,n)\text{-cca}}_{\mathsf{HEG}[\mathsf{GGen},\mathtt{high}]}(\mathcal{A}) \leq \mathrm{Adv}^{(m,n)\text{-gcdh}}_{\mathsf{GGen},\mathtt{high}}(\mathcal{B}) \ .$$

$\mathcal{B}$ runs adversary $\mathcal{A}$ once, uses its oracle DDH at most once per call to oracle RO, and performs some minor bookkeeping.

We conclude by sketching how $\mathcal{B}$ has to be modified for the cases of medium and low granularity. In the case of medium granularity $\mathcal{B}$'s challenge is of the form $(\boldsymbol{\mathcal{G}}, \boldsymbol{X}, \boldsymbol{Y})$, where all $\boldsymbol{\mathcal{G}}[i]$ describe the same group $\mathbb{G}$ of order p but with respect to different generators $\boldsymbol{g}[i]$. In this case $\mathcal{B}$ sets $par \leftarrow (\mathbb{G}, p)$ and $\boldsymbol{pk}[i] \leftarrow (\boldsymbol{g}[i], \boldsymbol{X}[i])$. In the case of low granularity $\boldsymbol{\mathcal{G}}$ consists of n independently sampled group descriptions. In this case $\mathcal{B}$ sets $par \leftarrow \bot$ and $\boldsymbol{pk}[i] \leftarrow (\boldsymbol{\mathcal{G}}[i], \boldsymbol{X}[i])$. It is easy to verify that in both cases the parameters and public keys are of the correct form. The rest of the proofs is analogous to the high-granularity case. □

E Efficiency of Hashed ElGamal in Different Parameter Settings

Below we illustrate how the scaling factors computed in Section 4.3 could be taken into account when choosing parameters for the Hashed-ElGamal KEM in the elliptic-curve setting dependent on m. In

Adversary $\mathcal{B}^{\mathrm{DDH}}(\mathcal{G}, \boldsymbol{X}, \boldsymbol{Y})$
00 initialize empty tables $T_{\mathrm{Dec}}, T_{\mathrm{RO}}, T_{\mathrm{DDH}}$
01 $\boldsymbol{K}^*[\cdot] \leftarrow \bot$; $\boldsymbol{c}^*[\cdot] \leftarrow \bot$
02 $\boldsymbol{Z'}[\cdot] \leftarrow \bot$
03 $\boldsymbol{b} \leftarrow_{\$} \{0,1\}^n$
04 $par \leftarrow \mathcal{G}[1]$
05 $\boldsymbol{pk} \leftarrow \boldsymbol{X}$
06 $(L, b') \leftarrow_{\$} \mathcal{A}^{\mathrm{SimEnc},\mathrm{SimDec},\mathrm{RO}}(par, \boldsymbol{pk})$
07 return $(L, \boldsymbol{Z'})$

Oracle SimEnc(i)
08 if $\boldsymbol{b}[i] = 1$:
09 $\quad \boldsymbol{K}^*[i] \leftarrow \mathrm{SimDec}(i, \boldsymbol{Y}[i])$
10 else: $\boldsymbol{K}^*[i] \leftarrow_{\$} \mathsf{KS}(par)$
11 $\boldsymbol{c}^*[i] \leftarrow \boldsymbol{Y}[i]$
12 return $(\boldsymbol{K}^*[i], \boldsymbol{c}^*[i])$

Oracle SimDec(i, c)
13 if $c = \boldsymbol{c}^*[i]$:
14 $\quad$ return $\bot$
15 if $T_{\mathrm{Dec}}[i, c] = \bot$:
16 $\quad \hat{Z} \leftarrow T_{\mathrm{DDH}}[i, \boldsymbol{X}[i], c]$
17 $\quad$ if $\hat{Z} \neq \bot$:
18 $\quad\quad T_{\mathrm{Dec}}[i, c] \leftarrow T_{\mathrm{RO}}[\boldsymbol{pk}[i], c, \hat{Z}]$
19 $\quad$ else: $T_{\mathrm{Dec}}[i, c] \leftarrow_{\$} \mathsf{KS}(par)$
20 return $T_{\mathrm{Dec}}[i, c]$

Oracle RO$(\boldsymbol{pk}[i], \hat{Y}, \hat{Z})$
21 if $T_{\mathrm{RO}}[\boldsymbol{pk}[i], \hat{Y}, \hat{Z}] \neq \bot$:
22 $\quad$ return $T_{\mathrm{RO}}[\boldsymbol{pk}[i], \hat{Y}, \hat{Z}]$
23 $T_{\mathrm{RO}}[\boldsymbol{pk}[i], \hat{Y}, \hat{Z}] \leftarrow_{\$} \mathsf{KS}(par)$
24 if $\mathrm{DDH}(i, \boldsymbol{X}[i], \hat{Y}, \hat{Z}) = 1$:
25 $\quad T_{\mathrm{DDH}}[i, \boldsymbol{X}[i], \hat{Y}] \leftarrow \hat{Z}$
26 $\quad$ if $\hat{Y} = \boldsymbol{Y}[i]$:
27 $\quad\quad \boldsymbol{Z'}[i] \leftarrow \hat{Z}$
28 $\quad$ if $T_{\mathrm{Dec}}[i, \hat{Y}] \neq \bot$:
29 $\quad\quad T_{\mathrm{RO}}[\boldsymbol{pk}[i], \hat{Y}, \hat{Z}] \leftarrow T_{\mathrm{Dec}}[i, \hat{Y}]$
30 return $T_{\mathrm{RO}}[\boldsymbol{pk}[i], \hat{Y}, \hat{Z}]$

Fig. 12. Adversary $\mathcal{B}$ against game $\mathrm{G}^{(m,n)\text{-gcdh}}_{\mathsf{GGen},\mathtt{high}}$. For simplicity we assume that all RO queries are of the form $(\boldsymbol{pk}[i], \hat{Y}, \hat{Z})$, where $\hat{Y}, \hat{Z}$ are elements of $par = \mathcal{G}[1]$.

practice HEG is typically instantiated with one out of a few standardized elliptic curves, which corresponds to the high granularity case. As seen in Section 4.3, this leads to a non-optimal scaling factor of $\sqrt{m}$. This motivates the question of whether it would be more efficient to compensate for this shortcoming by increasing the security parameter (i.e., the size of the used curve) or by switching to the low granularity variant with scaling factor m.

For concreteness, assume that 128 is the targeted single-instance security level of the KEM, i.e., breaking the $(1,1)$-CCA security of HEG takes time 2^{128}. For an optimally scaling KEM and a maximal number of m users this implies that the best attack against (m,m)-CCA should take at least time $2^{128} \cdot m$. Starting from this targeted multi-instance security level, we can compute the required group size p and corresponding key sizes of Hashed ElGamal for high and low granularity as follows: For high granularity we obtain

$$2^{128} \cdot m \approx \sqrt{p\,m} \ .$$

Solving for p yields $p \approx 2^{256} \cdot m$ leading to ciphertext and public-key sizes of $256 + \log m$ bits. On the other hand, for low granularity we get

$$2^{128} \cdot m \approx \sqrt{p} \cdot m \ ,$$

which leads to group sizes of 256 bits. However, in the low-granularity variant public keys not only consist of a single group element but also of the full group description, which for elliptic curves consists of 4 additional integers of bit length 256. Summing up, in order achieve full m-out-of-m multi-instance security starting from a single-instance security level of 128 bits we arrive at ciphertext and public-key sizes of

$$|c| = \begin{cases} 256 + \log m & \mathtt{gran} = \mathtt{high} \\ 256 & \mathtt{gran} = \mathtt{low} \end{cases} \quad \text{and} \quad |pk| = \begin{cases} 256 + \log m & \mathtt{gran} = \mathtt{high} \\ 1280 & \mathtt{gran} = \mathtt{low} \end{cases} \ ,$$

respectively. This shows that compensating for the non-optimal scaling of the high granularity variant of HEG leads to larger ciphertext sizes but substantially smaller keys compared to $\mathsf{HEG}[\mathsf{GGen}_{\mathbb{E}(\mathbb{F}_\ell)}, \mathtt{low}]$. (For comparison, if we assign 2^{20} keys to each inhabitant of Earth we get $\log m \approx 53$.) Additionally, this variant profits from faster key-generation since no elliptic curves have to be generated as part of the public key. The situation for efficiency of encapsulation and decapsulation is less clear: Formally, the larger group size in the high-granularity setting leads to slower group operations. On the other hand, the use of a fixed group allows for optimized implementations.

```
Adversary 𝒜((𝔾, p, g), X)
00 x[·] ← ⊥                          \\ list containing computed DLs
01 T[·] ← ⊥  \\ list containing elements with known DL
02 H ← g
03 T[H] ← 1
04 for i ∈ [2 .. √mp]:
05    H ← H · g                      \\ baby step
06    T[H] ← i
07 step ← H                          \\ H is g^√mp
08 for j ∈ [1 .. m]:                 \\ break the first m challenges
09    H ← X[j]
10    i ← 0
11    while T[H] = ⊥:
12       H ← H · step                \\ giant step
13       i ← i + 1
14    x[j] ← T[H] − i√mp
15 return x
```

Fig. 13. Baby-step giant-step algorithm for m DL instances. We redefine square root and division to round up to the next integer.

F Adversary Against (m, m)-DL in High Granularity

In this section we recall the baby-step giant-step algorithm for multiple instances and explicitly upper bound the number of group operations it needs to succeed by $2\sqrt{mp}$.

The algorithm (and its extension in the multi-instance setting) is well known in the literature. See for example [14, Section 9.2.4]. Here we give a concrete bound and a proof for completeness.

Theorem 7 formally states our result, where for concreteness the group is modeled as a generic group. The idea is as follows: In the baby-step phase, the adversary precomputes and stores the $\sqrt{mp}$ elements $g^1, g^2, g^3, \ldots, g^{\sqrt{mp}}$. In the giant-step phase, for each of the m DL challenges X it computes $Xg^{\sqrt{mp}}$, $Xg^{2\sqrt{mp}}$, $Xg^{3\sqrt{mp}}$, ... until it obtains one of the precomputed element. This latter phase stops before $\sqrt{p/m}$ computations, since after $\sqrt{p/m}$ steps of size $\sqrt{mp}$ the adversary loops through the whole group. The total number of operations is then $\sqrt{mp} + m\sqrt{p/m} = 2\sqrt{mp}$.

Theorem 7. *Let* $\mathsf{GGen}_{\mathsf{gg}}$ *be a group-generating algorithm that generates generic groups of prime size p, and let m be a positive integer. Then the baby-step giant-step algorithm, described as generic adversary $\mathcal{A}$ in Fig. 13, breaks* (m, m)-DL$[\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}]$ *with probability 1 and makes $q < 2\sqrt{mp}$ queries to the group-operation oracle.*

Proof. Consider adversary $\mathcal{A}$ described in Fig. 13.

Adversary $\mathcal{A}$ can be divided in two main phases. In the fist phase (lines 04 to 06), also called precomputation phase, it computes a list T containing the group elements $g^1, g^2, \ldots, g^{\sqrt{mp}}$ by multiplying each successive element by g, starting with g. (For the duration of the proof, we assume the square root function to round up its output to the nearest integer.) Next, $\mathcal{A}$ sets the variable *step* to the last computed element of T, that is, $g^{\sqrt{mp}}$. In the second phase (lines 08 to 14), also called search phase, $\mathcal{A}$ recovers each single DL from the challenges using the previous list. For each challenge entry X, $\mathcal{A}$ multiplies X by *step* until it obtains an element in T. Then it compute the DL as in line 14.

First we prove that adversary Fig. 13 succeeds with probability 1. Calling $X = g^x$ a challenge entry, $\mathcal{A}$ computes, in the second step, values of the form $X' = g^{x+i\sqrt{mp}}$, where i is the loop index. If X' belongs to list T, then we know that $X' = g^{T[X']}$ by definition of T. This means that $x + i\sqrt{mp} = T[X']$, which corresponds to the expression used to extract the DL in line 14.

Finally, we show that the algorithm makes $q < 2\sqrt{mp}$ queries. (And, in particular, that it terminates). The precomputation phase needs exactly $\sqrt{mp} - 1$ calls to the group-operation oracle (line 05). To count the number of group operations needed in the search phase, we give an upper bound on the number of iterations of the loop in lines 11 and 12. Let $X = g^x$, $x \in [1 .. p]$, be the challenge used in the loop. Applying integer division between $p - x$ and $\sqrt{mp}$, there exist two integer d, r such that $0 \le r < \sqrt{mp}$ and $p - x = d\sqrt{mp} + r$. Then, by $0 \le r < \sqrt{mp}$ we can deduce that $p < x + (d+1)\sqrt{mp} \le p + \sqrt{mp}$,

which implies that $Xg^{(d+1)\sqrt{mp}}$ belongs to T. Next, we assume $x > \sqrt{mp}$: Otherwise, the loop runs zero steps. Moreover, assuming $x > \sqrt{mp}$ (otherwise the loop runs either zero or, for $x = 0$, one steps) we get $d = (p - x - r)/\sqrt{mp} < p/\sqrt{mp} - 1$. (Note that we can assume $p/\sqrt{mp} > 1$: Otherwise the precomputation phase contains all group elements, and the number of queries of $\mathcal{A}$ is exactly $p-1 < \sqrt{mp}$.) Since $\mathcal{A}$ needs $d+1$ iterations of the while loop to compute $Xg^{(d+1)\sqrt{mp}}$, we conclude that each while loop runs less than $p/\sqrt{mp} > 0$ times. Summing up the queries including precomputation and all m challenges we get:

$$q < (\sqrt{mp} - 1) + m(p/\sqrt{mp}) = \sqrt{mp} + mp/\sqrt{mp} - 1 \ .$$

The bound in the statement follows since $\lceil x \rceil + x^2/\lceil x \rceil - 1 \leq \lceil x \rceil + x - 1 < 2x$ for $x > 0$. □

G Omitted Proofs of Section 6

G.1 Medium-Granularity (m, n)-GapCDH

In order to prove Corollary 4 we first present a simple reduction that describes how any adversary against (m, n)-GapCDH[$\mathsf{GGen}, \mathtt{med}$] can be used to break (m, n)-GapCDH[$\mathsf{GGen}, \mathtt{high}$]. Then we apply Corollary 3 to derive the concrete bound. We state the existence of a reduction from (m, n)-GapCDH in the medium-granularity setting to (m, n)-GapCDH in the high-granularity setting.

Lemma 11. *Let* GGen *be a group-generating algorithm that generates groups of at least size* p*, and let* m*,* n *be positive integers with* $m \leq n$*. Then for every adversary* $\mathcal{A}$ *against* (m, n)-GapCDH[$\mathsf{GGen}, \mathtt{med}$] *there exists an adversary* $\mathcal{B}$ *against* (m, n)-GapCDH[$\mathsf{GGen}, \mathtt{high}$] *such that*

$$\mathrm{Adv}^{(m,n)\text{-gcdh}}_{\mathsf{GGen},\mathtt{high}}(\mathcal{B}) \geq \mathrm{Adv}^{(m,n)\text{-gcdh}}_{\mathsf{GGen},\mathtt{med}}(\mathcal{A}) \ .$$

Moreover, calling q_{DDH} *the number of queries of* $\mathcal{A}$ *to* DDH*, adversary* $\mathcal{B}$ *makes at most* $2(4n+q_{\mathrm{DDH}})(\log p+1)$ *group operations in addition to those made by* $\mathcal{A}$*, and the same amount of queries to* DDH*.*

Proof. Adversary $\mathcal{B}$ works as follows. $\mathcal{B}$ receives a group description $\mathcal{G} = (\mathbb{G}, p, g)$ and some n-size vectors $\boldsymbol{X} = g^{\boldsymbol{x}}$, $\boldsymbol{Y} = g^{\boldsymbol{y}}$. It generates a uniform vector, $\boldsymbol{r} \leftarrow_{\$} (\mathbb{Z}_p \setminus 0)^n$, and computes $\boldsymbol{g'} \leftarrow g^{\boldsymbol{r}}$; $\boldsymbol{X'}[\cdot] \leftarrow \boldsymbol{X}[\cdot]^{\boldsymbol{r}[\cdot]}$; $\boldsymbol{Y'}[\cdot] \leftarrow \boldsymbol{Y}[\cdot]^{\boldsymbol{r}[\cdot]}$. The computed elements correspond to the new group generators and the new challenges for $\mathcal{A}$. Queries to DDH on input of index i and group elements $\hat{X}, \hat{Y}, \hat{Z}$ are answered as follows. $\mathcal{B}$ queries its own oracle on input $\hat{X}, \hat{Y}^{1/\boldsymbol{r}[i]}, \hat{Z}$ and forwards the output to $\mathcal{A}$.

We prove that the oracle simulation is correct. Let $\hat{x}$, $\hat{y}$, $\hat{z}$ be such that $\hat{X} = g^{\hat{x}}$, $\hat{Y} = g^{\hat{y}}$, and $\hat{Z} = g^{\hat{z}}$ respectively. Since $\boldsymbol{g'}[i] = g^{\boldsymbol{r}[i]}$, $\mathcal{A}$ needs to receive output 1 if and only if

$$\left(g^{\boldsymbol{r}[i]}\right)^{\frac{\hat{x}}{\boldsymbol{r}[i]}\frac{\hat{y}}{\boldsymbol{r}[i]}} = \left(g^{\boldsymbol{r}[i]}\right)^{\frac{\hat{z}}{\boldsymbol{r}[i]}} \ ,$$

or, equivalently, $g^{\hat{x}\hat{y}/\boldsymbol{r}[i]} = g^{\hat{x}}$. This corresponds to a DDH query with respect to group generator g and input $\hat{X}, \hat{Y}^{1/\boldsymbol{r}[i]}, \hat{Z}$.

Next, we show that if $\mathcal{A}$ is successful then $\mathcal{B}$ is. If $\mathcal{A}$ is successful, then its output is a vector $\boldsymbol{W'}$ and a set $L = \{l_1, \ldots, l_m, \ldots\}$ such that $\boldsymbol{W'}[l_i] = (\boldsymbol{g'}[l_i])^{\boldsymbol{x}[l_i]\boldsymbol{y}[l_i]}$ for every i. If $\mathcal{B}$ sets $\boldsymbol{W}[\cdot] \leftarrow \boldsymbol{W'}[\cdot]^{1/\boldsymbol{r}[\cdot]}$ then $\boldsymbol{W}[l_i] = g^{\boldsymbol{x}[l_i]\boldsymbol{y}[l_i]}$ for every i. $(L, \boldsymbol{W})$ is the final output of $\mathcal{B}$, which wins the high granularity game.

To conclude the proof, we count the additional group queries made by $\mathcal{B}$. The group-operation oracle is used when creating the new generators (n), challenges ($2n$), answering DDH queries (q_{DDH}), and computing the output entries (at most n). By using square-and-multiply we can upper bound the amount of queries by $2(4n + q_{\mathrm{DDH}})(\log p + 1)$. □

Proof (Corollary 4). Combining Lemma 11, and Corollary 3, and observing that the reduction makes additional $2(4n + q_{\mathrm{DDH}})(\log p + 1) + 2n(m + 2)(\log p + 1)$ group operations:

$$\begin{aligned}\mathrm{Adv}^{(m,n)\text{-gdl}}_{\mathsf{GGen}_{\mathsf{gg}},\mathtt{med}}(\mathcal{A}) &\leq \mathrm{Adv}^{(m,n)\text{-gcdh}}_{\mathsf{GGen}_{\mathsf{gg}},\mathtt{high}}(\mathcal{B}) \leq 2^m \mathrm{Adv}^{(m,m)\text{-gdl}}_{\mathsf{GGen}_{\mathsf{gg}},\mathtt{high}}(\mathcal{C}) \\ &\leq \left(\frac{2}{p}\right)^m + \frac{1}{2}\left(\frac{e(q + 2(q_{\mathrm{DDH}} + n(m+6))(\log p + 1) + m + 1)^2 + 2eq_{\mathrm{DDH}}}{mp}\right)^m \ .\end{aligned}$$

The previous expression can be simplified by noticing that, for the parameters we consider, $2n(m + 6)(\log p + 1) + m + 1 \leq 30mn \log p$, $e(2q_{\mathrm{DDH}})^2 \geq 2eq_{\mathrm{DDH}}$, $2q_{\mathrm{DDH}}(\log p + 1) \leq 6q_{\mathrm{DDH}} \log p$, and $2\sqrt[m]{2} \leq 4 \leq (30mn \log p)^2/m$. □

G.2 Low-Granularity (m, n)-GapCDH

Proof (Theorem 6). Let $\mathcal{A}$ be a generic adversary against (m, n)-GapCDH[$\mathsf{GGen}_{\mathsf{gg}}$, $\mathtt{low}$], and let $\boldsymbol{W}$ be the random variable representing the (unique) solution vector to all challenges. We model the n groups generated by $\mathsf{PGen}[\mathtt{low}]$ as independent generic groups of size $\boldsymbol{p}$, where $\boldsymbol{p}[i] \geq p$ for every $i \in [1\,..\,n]$. Adversary $\mathcal{A}$ does not know the solution $\boldsymbol{W}[i]$ of the i-th challenge unless it is returned by the i-th group-operation oracle. By Corollary 3, for $m = n = 1$ we know that:

$$\Pr\left[\mathcal{A} \text{ computes } \boldsymbol{W}[i]\right] \leq \frac{2e(q_i + 12\log p)^2 + 4eq_{\mathrm{DDH}}}{p} \leq 4e\frac{q_i^2}{p} \quad , \tag{25}$$

where the last inequality follows since q_i is large: $q_i \geq 60\log p$, which implies $(q_i + 12\log p) \leq 3/2q_i^2$, and $4q_i^2 \geq q_{\mathrm{DDH}}$. Adversary $\mathcal{A}$ wins exclusively if it can identify at least m CDH solutions. Let $L = \{l_1, \ldots, l_m, \ldots\}$ be the solution indices returned by $\mathcal{A}$. For the purpose of the proof, we define the set of indices $S = \{i_1, \ldots, i_m\}$ such that $q_{i_1} \geq \ldots \geq q_{i_m} \geq q_i$ for every $i \in [1\,..\,n] \setminus S$. We can bound the winning probability as:

$$\begin{aligned}
\Pr[\mathrm{G}^{(m,n)\text{-gcdh}}_{\mathsf{GGen}_{\mathsf{gg}},\mathtt{low}}(\mathcal{A})] &\leq \Pr\left[\mathcal{A} \text{ computes } \boldsymbol{W}[\{l_1, \ldots, l_m\}]\right] \\
&\leq \prod_{i=1}^{m} \Pr\left[\mathcal{A} \text{ computes } \boldsymbol{W}[l_i]\right] \\
&\leq \prod_{i \in S} \left(4\frac{q_i^2}{p}\right) \leq \left(\frac{4e}{p}\left(\frac{1}{m}\sum_{i \in S} q_i\right)^2\right)^m \leq \left(\frac{4eq^2}{m^2 p}\right)^m \quad .
\end{aligned}$$

To prove this bound we used, in order, the definition of $\mathrm{G}^{(m,n)\text{-gcdh}}_{\mathsf{GGen}_{\mathsf{gg}},\mathtt{low}}$, the independence of each generic group, Eq. (25) and the fact that, after reordering, $q_{i_j} \leq q_{l_j}$ for every possible L and $j \in [1\,..\,m]$, and the inequality of arithmetic and geometric means, that is, for every nonnegative x_i such that $x_1 + \ldots + x_m = x$:

$$\prod_{i=1}^{m} x_i \leq \left(\frac{x}{m}\right)^m \quad . \qquad \square$$