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

Benedikt Auerbach[1](✉), Federico Giacon[2](✉), and Eike Kiltz[3]

[1] IST Austria, Klosterneuburg, Austria
benedikt.auerbach@ist.ac.at
[2] Gnosis Service GmbH, Berlin, Germany
federico.giacon@rub.de
[3] Ruhr-Universität Bochum, Bochum, Germany
eike.kiltz@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 describe how 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 complexity of 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.

A. Canteaut and Y. Ishai (Eds.): EUROCRYPT 2020, LNCS 12107, pp. 475–506, 2020.
https://doi.org/10.1007/978-3-030-45727-3_16

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. [7]. In the (security-wise) best possible scenario, running an m-out-of-n multi-instance attack is m times more difficult compared to a (standard) single-instance attack. 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 give a quantitative answer to this question, 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 studied 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 [2,19], 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

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

Table 1. Shared public system parameters and individual public keys for schemes $\mathsf{HEG}[\mathsf{GGen}_{\mathbb{E}(\mathbb{F}_\ell)}, \mathtt{gran}]$ and $\mathsf{HEG}[\mathsf{GGen}_{\mathbb{F}_\ell^*}, \mathtt{gran}]$ 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.

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

instances. Examples are factoring [11] and computing discrete logarithms in the finite-field [4,5] and elliptic-curve [18,20,22] 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), also known as 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 three different granularity settings (low, medium, and high), leading to six 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 $\mathsf{HEG}[\mathsf{GGen}_{\mathbb{F}_\ell^*}, \mathtt{high}]$, which offer 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.

1.1 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. [8] 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 for PKE [7] is different: Most importantly, multi-user security involves 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. [8] argue that this is a natural metric because its well-known "sensitivity" means that as long as at least one of the challenge bits looks random to the adversary so does their XOR. They further argue that other possible winning conditions such as using AND[2] are less natural and lead to inconsistencies. We refer to Bellare et al. [8] 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 m and n. The natural results we are able to show in this regard (among others, the intuitive idea that a single-instance adversary of advantage ϵ and running time t can be extended to an m-out-of-n adversary of advantage ϵ^m and running time mt; see Theorem 1) give us further confidence on the significance of the chosen multi-instance security definition, and enable us to present a formally sound definition of the scaling factor.

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

[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 $\mathrm{Adv}^{(m,n)\text{-cpa}}_{\mathsf{PKE}} = \Pr[\bigwedge_{i \in I} b[i] = b'[i]] - 1/2^m$.

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 $\delta \geq 0$.

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	$\begin{cases} L_\ell(1/3, \delta) & 0 \leq \delta < 0.105 \\ L_\ell(1/3, 0.105) & 0.105 \leq \delta < 0.368 \\ L_\ell(1/3, -0.263 + \delta) & 0.368 \leq \delta \end{cases}$

require either an unproven complexity assumption or a restricted model of computation. We rely on the generic group model [28] 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 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. In both cases however, HEG scales better in the low-granularity case. Concretely, Hashed ElGamal over 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 $L_\ell(1/3, c) := \exp((c + o(1))(\log \ell)^{1/3}(\log\log \ell)^{2/3})$. 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, the point starting from which the cumulative cost of breaking m individual instances outweighs the cost of the precomputation. 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 empirical results [2, Table 2]. For low granularity and for up to $L_\ell(1/3, 0.105)$ instances, $\mathsf{HEG}[\mathsf{GGen}_{\mathbb{F}^*_\ell}, \mathtt{low}]$ scales optimally. For $L_\ell(1/3, 0.105) \leq m \leq L_\ell(1/3, 0.368)$, the scaling factor is roughly constant, and for larger numbers of instances, it scales linearly with slope $L_\ell(1/3, -0.263)$ which is far larger than the slope in the case of medium or high granularity.

Summing up, Hashed ElGamal instantiated with elliptic curve groups shows a better scaling behavior than the corresponding instantiation in the finite-field setting. Further, in both cases switching from the high granularity setting to the medium granularity setting does not improve the scaling behavior, while the

Table 3. Example values of scaling factor $\mathrm{SF}^{(m,m)\text{-cca}}_{\mathsf{HEG}[\mathsf{GGen},\mathtt{gran}]}$ for different values of m and ℓ, $\mathsf{GGen} \in \{\mathsf{GGen}_{\mathbb{E}(\mathbb{F}_\ell)}, \mathsf{GGen}_{\mathbb{F}^*_\ell}\}$, and $\mathtt{gran} \in \{\mathtt{high}, \mathtt{med}, \mathtt{low}\}$.

		Elliptic curve		Finite field	
m	ℓ	`high`, `med`	`low`	`high`, `med`	`low`
2^{20}	512	2^{10}	2^{20}	1.21	$2^{11.26}$
	1024	2^{10}	2^{20}	1.00	$2^{8.26}$
	2048	2^{10}	2^{20}	1.00	$2^{6.64}$
2^{30}	512	2^{15}	2^{30}	$2^{7.73}$	$2^{21.26}$
	1024	2^{15}	2^{30}	1.85	$2^{18.13}$
	2048	2^{15}	2^{30}	1.00	$2^{14.02}$

use of individual groups, i.e., low-granularity parameters does. To illustrate our findings we provide example values of the scaling factor for different numbers of instances m and prime sizes ℓ in Table 3.

While our results imply that the use of low-granularity parameters is preferable with respect to security scaling, we stress that generating cryptographically secure groups is a hard and error prone process. Delegating this task to the the individual user as part of the key generation might actually have a negative impact on the scheme's security in practice. Further, the use of individual groups negatively impacts the efficiency of the scheme, as key generation requires the sampling of secure groups, and key-sizes increase.

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 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. *The* (m, n)-CCA *security of* $\mathsf{HEG}[\mathsf{GGen}, \mathtt{gran}]$ *is tightly implied by the hardness of* (m, n)-GapCDH$[\mathsf{GGen}, \mathtt{gran}]$.

The theorem (described formally in Sect. 4) is a somewhat straightforward generalization of the single-instance case [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 [4, 5] 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 [5] (high/medium granularity) and the DLOG factory [4] (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 prove the following generic-group lower bounds for (m, n)-GapCDH$[\mathsf{GGen}_{\mathsf{gg}}, \mathtt{gran}]$ in different granularities, where $\mathsf{GGen}_{\mathsf{gg}}$ generates a generic group [28] of prime order p, and the granularity $\mathtt{gran}$ determines how much information about the used group is shared amongst the challenge instances (see Table 4).

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

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

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

The lower bounds on $\mathrm{SF}^{m,n}$ for $\mathsf{HEG}[\mathsf{GGen}_{\mathbb{E}(\mathbb{F}_\ell)}, \mathtt{gran}]$ 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

We consider multi-instance variants of three different problems: the discrete logarithm problem ((m, n)-DL$[\mathsf{GGen}_{\mathsf{gg}}, \mathtt{gran}]$), the gap discrete logarithm problem ((m, n)-GapDL$[\mathsf{GGen}_{\mathsf{gg}}, \mathtt{gran}]$), and the gap computational Diffie-Hellman problem ((m, n)-GapCDH$[\mathsf{GGen}_{\mathsf{gg}}, \mathtt{gran}]$) in different granularities, see Table 4.

We now discuss the complexity column of Table 4. It is well known that the running time of solving (m, n)-DL$[\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}]$ is $\Theta(\sqrt{mp})$, the lower bound being in the generic group model [29,30], the matching upper bound stemming from a concrete generic algorithm [22]. It is not hard to see that the bounds on (m, n)-DL$[\mathsf{GGen}_{\mathsf{gg}}, \mathtt{med}]$ are basically the same because the generators g_i can be viewed as "high-granularity instances" g^{x_j}. Concerning low granularity, it is noteworthy to mention the bound for the case $m = n$ by Garay et al. [17]. Using different techniques, we are able to improve their bound from $\sqrt{mp}$ to $m\sqrt{p}$. In addition, our bound also holds in the case $m < n$ and in the gap setting.

Our **first main technical result** (Corollary 1) is a non-trivial extension of Yun's generic lower bound [30] to the gap setting, i.e., a new lower bound of $\Omega(\sqrt{mp})$ on solving (m, m)-GapDL$[\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}]$. Based on this result, we also deduce bounds in the case of medium and low granularity.

Table 4. Definition and generic-group complexity of problems (m, n)-DL[$\mathsf{GGen}, \mathtt{gran}$], (m, n)-GapDL[$\mathsf{GGen}, \mathtt{gran}$], and (m, n)-GapCDH[$\mathsf{GGen}, \mathtt{gran}$], where $\mathtt{gran}$ belongs to $\{\mathtt{high}, \mathtt{med}, \mathtt{low}\}$. $\mathbb{G}$ 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-out-of-n problem	Given	Break m out of	Gap?	Complexity	Ref.
DL[$\mathsf{GGen}, \mathtt{high}$]	$\mathbb{G}, p, g, g^{x_1}, \ldots, g^{x_n}$	$x_1, \ldots, x_n$	–	$\Theta(\sqrt{mp})$	[22, 29, 30]
DL[$\mathsf{GGen}, \mathtt{med}$]	$\mathbb{G}, p, g_1, g_1^{x_1}, \ldots, g_n, g_n^{x_n}$	$x_1, \ldots, x_n$	–	$\Theta(\sqrt{mp})$	full version [3]
DL[$\mathsf{GGen}, \mathtt{low}$]	$\mathbb{G}_1, p_1, g_1, g_1^{x_1}, \ldots, \mathbb{G}_n, p_n, g_n, g_n^{x_n}$	$x_1, \ldots, x_n$	–	$\Theta(m\sqrt{p})$	full version [3]
GapDL[$\mathsf{GGen}, \mathtt{high}$]	$\mathbb{G}, p, g, g^{x_1}, \ldots, g^{x_n}$	$x_1, \ldots, x_n$	✓	$\Theta(\sqrt{mp})$	§5.2
GapDL[$\mathsf{GGen}, \mathtt{med}$]	$\mathbb{G}, p, g_1, g_1^{x_1}, \ldots, g_n, g_n^{x_n}$	$x_1, \ldots, x_n$	✓	$\Theta(\sqrt{mp})$	full version [3]
GapDL[$\mathsf{GGen}, \mathtt{low}$]	$\mathbb{G}_1, p_1, g_1, g_1^{x_1}, \ldots, \mathbb{G}_n, p_n, g_n, g_n^{x_n}$	$x_1, \ldots, x_n$	✓	$\Theta(m\sqrt{p})$	full version [3]
GapCDH[$\mathsf{GGen}, \mathtt{high}$]	$\mathbb{G}, p, g, g^{x_1}, g^{y_1}, \ldots, g^{x_n}, g^{y_n}$	$g^{x_1 y_1}, \ldots, g^{x_n y_n}$	✓	$\Theta(\sqrt{mp})$	§6.1
GapCDH[$\mathsf{GGen}, \mathtt{med}$]	$\mathbb{G}, p, g_1, g_1^{x_1}, g_1^{y_1}, \ldots, g_n, g_n^{x_n}, g_n^{y_n}$	$g_1^{x_1 y_1}, \ldots, g_n^{x_n y_n}$	✓	$\Theta(\sqrt{mp})$	§6.2
GapCDH[$\mathsf{GGen}, \mathtt{low}$]	$\mathbb{G}_1, p_1, g_1, g_1^{x_1}, g_1^{y_1}, \ldots, \mathbb{G}_n, p_n, g_n, g_n^{x_n}, g_n^{y_n}$	$g_1^{x_1 y_1}, \ldots, g_n^{x_n y_n}$	✓	$\Theta(m\sqrt{p})$	§6.3

Our **second main technical result** (Theorem 4) states that, in high granularity, the (m, m)-GapDL and the (m, n)-GapCDH problems are essentially equally hard in the algebraic group model [16], 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[$\mathsf{GGen}_{\mathrm{gg}}, \mathtt{high}$]. 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 DDH(g^x, g^y, g^z) by querying Eval$(f := X_1 X_2 - X_3, g^x, g^y, g^z)$. 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[$\mathsf{GGen}_{\mathrm{gg}}, \mathtt{high}$]) 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 [30] for the standard multi-instance DL problem. In line with Yun's approach, we define the *search-by-hypersurface* problem in dimension m (m-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 $\mathbb{Z}_p[X_1, \ldots, X_m]$ of degree at most d. Notably, Yun's *search-by-hyperplane-queries* problem in dimension m is equivalent to m-SHS$_1$. We stress that the more general case of $d \geq 1$ requires significantly different arguments from commutative algebra/algebraic geometry, compared to the linear algebra argument used for the DL bound.

We show that any generic adversary against (m, m)-d-PolyDL[$\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}$] 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 commutative algebra. 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[$\mathsf{GGen}, \mathtt{high}$] Hardness Implies (m, n)-GapCDH[$\mathsf{GGen}, \mathtt{high}$]. The algebraic group model [16] is a technique used to extend existing bounds in the generic group model to different problems by means of generic reductions. Our second technical result (Theorem 4) presents a generic 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:

$$\mathrm{Adv}_{\mathtt{high}}^{(m,n)\text{-gcdh}} \overset{\text{Th. 4}}{\leq} 2^m \cdot \mathrm{Adv}_{\mathtt{high}}^{(m,m)\text{-gdl}} \overset{\text{Cor. 1}}{\lesssim} 2^m \left(\frac{q^2 + q_{\mathrm{DDH}}}{mp}\right)^m \approx \left(\frac{2q^2}{mp}\right)^m,$$

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[$\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}$] bound. More importantly, by the above bound one requires $q \geq \Omega(\sqrt{mp})$ generic-group operations to break (m, n)-GapCDH[$\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}$] with overwhelming advantage.

A natural approach to tackle the proof of Theorem 4 would be to adapt the single-instance proof presented by Fuchsbauer et al. [16] to the multi-instance setting. Following this strategy in a reduction, however, one would need to argue about the size of the solution set of a multivariate system of quadratic equations. In this work we employ significantly different proof techniques.

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.

1.3 Related Work and Future Directions

Related Work. Multi-instance security in the sense of breaking m out of m instances was first formally considered in the setting of symmetric encryption by Bellare et al. [8]. We point out that the term is sometimes also used to describe multi-user, multi-challenge generalizations of single-instance security notions [21].

The (single-instance) GapCDH problem was introduced by Okamoto and Pointcheval [25]. Boneh et al. [12] and Rupp et al. [26] provide frameworks in the generic-group model that can be used to derive generic-group lower bounds on the hardness of many single-instance problems, gapCDH amongst others. The generic hardness of (m, m)-DL in the high-granularity setting was first analyzed by Yun [30], the result later generalized to (m, n)-DL by Ying and Kunihiro [29]. Kuhn and Struik [22], and Fouque et al. [15] give generic algorithms matching the lower bounds. The first bound for (m, m)-DL in the low granularity setting was derived by Garay et al. [17]. The algebraic-group model was introduced by Fuchsbauer et al. [16]. Mizuide et al. [24] provide a framework that can be used to reduce single-instance CDH-type problems to the discrete-logarithm problem in the algebraic-group model.

Bartusek et al. [6] and Sadeghi et al. [27] discuss differences between DL-type assumptions depending on whether the used group and group generator are fixed or sampled at random. We stress that in this work groups and group generators, while potentially shared amongst different users, are sampled at the beginning of the game and hence part of its probability space.

Future Directions. Corrigan-Gibbs and Kogan [14] 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. [13] 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 [9]) 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* [10]. 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 the literature [23,28]. 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. [16], this idea gives a handy technique for carrying over generic lower bounds through generic reductions, as seen in the following lemma.

Lemma 1. **([16, Lemma 1]).** *Let α, Δ be constants and let $\mathcal{R}$ be a generic reduction $\mathcal{R}$ from game* G_1 *to* G_0. *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 Sect. 3.1, in Sect. 3.2 we consider the relation between security notions for varying m and n. In Sect. 3.3 we define the scaling factor, which measures how well the security of KEMs scales with the number of users. Finally, in Sect. 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. [8] for symmetric encryption schemes. 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 $L \subseteq [1\mathinner{..}n]$ 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 with 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 $i \in [1\mathinner{..}n]$ 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 $L \subseteq [1\mathinner{..}n]$ 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

```
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 $i \in [1\,..\,n]$:
    $(\boldsymbol{pk}[i], \boldsymbol{sk}[i]) \leftarrow_\$ \mathsf{Gen}(par)$
$(L, b') \leftarrow_\$ \mathcal{A}^{\mathrm{Enc}}(par, \boldsymbol{pk})$              \\ $(m,n)$-CPA
$(L, b') \leftarrow_\$ \mathcal{A}^{\mathrm{Enc},\mathrm{Dec}}(par, \boldsymbol{pk})$         \\ $(m,n)$-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 $L \subseteq [1\,..\,n]$.

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. [7]. We describe the relationship between multi-user security and $(1,n)$-CCA in detail in the full version of the paper [3].

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. Its proof is in the full version of the paper [3]

Theorem 1. *Let m, n, m', n' be positive integers such that $m \leq n$, $m' \leq n'$, 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 $m' \leq m$ and $m'n \leq mn'$ then $\mathcal{B}$ has roughly the same running time of $\mathcal{A}$ and*
$$\mathrm{Adv}^{(m',n')\text{-cpa}}_{\mathsf{KEM}}(\mathcal{B}) \geq \frac{1}{2}\mathrm{Adv}^{(m,n)\text{-cpa}}_{\mathsf{KEM}}(\mathcal{A}).$$
Additionally, if $n'-m' \geq n-m$ then the reduction does not lose the factor $1/2$.

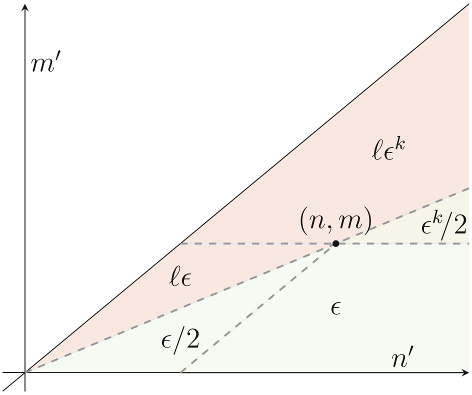

Fig. 2. Relations between (m', n')-CPA and (m, n)-CPA security. Given $\mathcal{A}$ against (m, n)-CPA with advantage ϵ, one can build $\mathcal{B}$ against (m', n')-CPA with advantage as shown in figure, depending on its position on the plane. The constants in the figure are $k = \lceil m'/m \rceil$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

2. *If $m' \leq m$ and $m'n > mn'$ then $\mathcal{B}$ has roughly the same running time of $\mathcal{A}$ and*
$$\mathrm{Adv}_{\mathsf{KEM}}^{(m',n')\text{-cpa}}(\mathcal{B}) \geq \frac{1}{2}\binom{n'}{m'}\binom{\lceil nm'/m \rceil}{m'}^{-1}\mathrm{Adv}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}(\mathcal{A}).$$
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*
$$\mathrm{Adv}_{\mathsf{KEM}}^{(m',n')\text{-cpa}}(\mathcal{B}) \geq \frac{1}{2}\left(\mathrm{Adv}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}(\mathcal{A})\right)^k.$$
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*
$$\mathrm{Adv}_{\mathsf{KEM}}^{(m',n')\text{-cpa}}(\mathcal{B}) \geq \frac{1}{2}\binom{n'}{m'}\binom{\lceil nm'/m \rceil}{m'}^{-1}\left(\mathrm{Adv}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}(\mathcal{A})\right)^k.$$

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 define the scaling factor of key-encapsulation mechanisms. To be able to give an intuitive and accessible definition we treat the running time and advantages of adversaries as if they were elements of $\mathbb{R}$ and $[0, 1]$ respectively. A formal definition that takes the asymptotic nature of running time and advantage into account as well as rigorous proofs for the bounds on the scaling factor derived in this section can be found in the full version of the paper [3]. We start with a definition for adversaries succeeding with advantage 1 and afterwards give a generalized version for arbitrary advantages.

We fix a computational model that associates each adversary $\mathcal{A}$ with its running time. Let $\mathrm{MinTime}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}$ be the minimal time T for which there exists an adversary $\mathcal{A}$ that runs in at most time T and achieves advantage $\mathrm{Adv}_{\mathsf{KEM}}^{(m,n)\text{-cpa}}(\mathcal{A}) = 1$.

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

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

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

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

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 choose 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 the full version of the paper [3]. 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:

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

The definition given above exclusively considers adversaries that achieve advantage 1. This definition generalizes 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, the advantage-dependent scaling factor for advantage ε is defined as

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

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 $\mathrm{G}^{(m,n)\text{-gdl}}_{\mathsf{GGen},\mathtt{gran}}(\mathcal{A})$ of Fig. 3 associated with 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 $L \subseteq [1 \mathinner{..} n]$ 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 $i \in [1 \mathinner{..} n]$ 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 $\mathrm{G}^{(m,n)\text{-gdl}}_{\mathsf{GGen},\mathtt{gran}}(\mathcal{A})$ as

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

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 $i \in [1 \mathinner{..} n]$. 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.

Formally, for $m, n \in \mathbb{N}$ s.t. $1 \leq m \leq n$ consider game $\mathrm{G}^{(m,n)\text{-gcdh}}_{\mathsf{GGen},\mathtt{gran}}(\mathcal{A})$ of Fig. 4 associated with 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 $\mathbb{Z}_{\boldsymbol{p}[i]}$.

```
Games $\mathrm{G}^{(m,n)\text{-gdl}}_{\mathsf{GGen},\mathtt{gran}}(\mathcal{A})$
$\boldsymbol{\mathcal{G}} \leftarrow_\$ \mathsf{PGen}[\mathtt{gran}]$
$\boldsymbol{x}[\cdot] \leftarrow_\$ \mathbb{Z}_{\boldsymbol{p}[\cdot]}$; $\boldsymbol{X}[\cdot] \leftarrow \boldsymbol{g}[\cdot]^{\boldsymbol{x}[\cdot]}$
$(L, \boldsymbol{x}') \leftarrow_\$ \mathcal{A}^{\mathrm{DDH}}(\boldsymbol{\mathcal{G}}, \boldsymbol{X})$
if $|L| < m$: return 0
if $\boldsymbol{x}'[L] = \boldsymbol{x}[L]$: return 1
else: return 0

Oracle $\mathrm{DDH}(i, \hat{X}, \hat{Y}, \hat{Z})$
parse $\hat{X}, \hat{Y}$ as $\boldsymbol{g}[i]^{\hat{x}}, \boldsymbol{g}[i]^{\hat{y}}$
if $\boldsymbol{g}[i]^{\hat{x}\hat{y}} = \hat{Z}$:
    return 1
else: return 0

Procedure $\mathsf{PGen}[\mathtt{high}]$
$\mathcal{G} = (\mathbb{G}, p, g) \leftarrow_\$ \mathsf{GGen}$
$\boldsymbol{\mathcal{G}}[\cdot] \leftarrow \mathcal{G}$
return $\boldsymbol{\mathcal{G}}$

Procedure $\mathsf{PGen}[\mathtt{med}]$
$(\mathbb{G}, p, g) \leftarrow_\$ \mathsf{GGen}$
$\boldsymbol{g} \leftarrow_\$ (\mathbb{G} \setminus \{1\})^n$
$\boldsymbol{\mathcal{G}}[\cdot] \leftarrow (\mathbb{G}, p, \boldsymbol{g}[\cdot])$
return $\boldsymbol{\mathcal{G}}$

Procedure $\mathsf{PGen}[\mathtt{low}]$
$\boldsymbol{\mathcal{G}}[\cdot] \leftarrow_\$ \mathsf{GGen}$
return $\boldsymbol{\mathcal{G}}$
```

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 gap discrete logarithm problem.

```
Game $\mathrm{G}^{(m,n)\text{-gcdh}}_{\mathsf{GGen},\mathtt{gran}}(\mathcal{A})$
$\boldsymbol{\mathcal{G}} \leftarrow_\$ \mathsf{PGen}[\mathtt{gran}]$
$\boldsymbol{x}[\cdot] \leftarrow_\$ \mathbb{Z}_{\boldsymbol{p}[\cdot]}$; $\boldsymbol{X}[\cdot] \leftarrow \boldsymbol{g}[\cdot]^{\boldsymbol{x}[\cdot]}$
$\boldsymbol{y}[\cdot] \leftarrow_\$ \mathbb{Z}_{\boldsymbol{p}[\cdot]}$; $\boldsymbol{Y}[\cdot] \leftarrow \boldsymbol{g}[\cdot]^{\boldsymbol{y}[\cdot]}$
$\boldsymbol{Z}[\cdot] \leftarrow \boldsymbol{g}[\cdot]^{\boldsymbol{x}[\cdot]\boldsymbol{y}[\cdot]}$
$(L, \boldsymbol{Z}') \leftarrow_\$ \mathcal{A}^{\mathrm{DDH}}(\boldsymbol{\mathcal{G}}, \boldsymbol{X}, \boldsymbol{Y})$
if $|L| < m$: return 0
if $\boldsymbol{Z}[L] = \boldsymbol{Z}'[L]$: return 1
else: return 0

Oracle $\mathrm{DDH}(i, \hat{X}, \hat{Y}, \hat{Z})$
parse $\hat{X}, \hat{Y}$ as $\boldsymbol{g}[i]^{\hat{x}}, \boldsymbol{g}[i]^{\hat{y}}$
if $\boldsymbol{g}[i]^{\hat{x}\hat{y}} = \hat{Z}$:
    return 1
else: return 0
```

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.

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 $L \subseteq [1 \mathinner{..} n]$ 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

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

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 Sect. 4.1, in Sect. 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 Sect. 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 $\mathtt{gran} \in \{\mathtt{high}, \mathtt{med}, \mathtt{low}\}$.

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 (m, n)-GapCDH tightly reduces to the security against chosen-ciphertext attacks of HEG in the multi-instance setting for the corresponding granularity[3]. Its proof is a generalization of the single-instance version [1] and can be found in the full version of the paper [3].

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 $\mathsf{HEG}[\mathsf{GGen}, \mathtt{gran}]$ 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$[\mathsf{GGen}, \mathtt{gran}]$ problem is hard, then*

[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.

```
gran = high

Alg. Par[high]
  $\mathcal{G} = (\mathbb{G}, p, g) \leftarrow_\$ \mathsf{GGen}$
  $par \leftarrow \mathcal{G}$
  return $par$

Alg. Gen[high]($par$)
  $x \leftarrow_\$ \mathbb{Z}_p$; $X \leftarrow g^x$
  $pk \leftarrow X$; $sk \leftarrow x$
  return $(pk, sk)$

gran = med

Alg. Par[med]
  $\mathcal{G} = (\mathbb{G}, p, g) \leftarrow_\$ \mathsf{GGen}$
  $par \leftarrow (\mathbb{G}, p)$
  return $par$

Alg. Gen[med]($par$)
  $g \leftarrow_\$ \mathbb{G} \setminus \{1\}$
  $x \leftarrow_\$ \mathbb{Z}_p$; $X \leftarrow g^x$
  $pk \leftarrow (g, X)$; $sk \leftarrow (g, x)$
  return $(pk, sk)$

gran = low

Alg. Par[low]
  $par \leftarrow \perp$
  return $par$

Alg. Gen[low]($par$)
  $\mathcal{G} = (\mathbb{G}, p, g) \leftarrow_\$ \mathsf{GGen}$
  $x \leftarrow_\$ \mathbb{Z}_p$; $X \leftarrow g^x$
  $pk \leftarrow (\mathcal{G}, X)$; $sk \leftarrow (\mathcal{G}, x)$
  return $(pk, sk)$

Alg. Enc($par, pk$)
  $y \leftarrow_\$ \mathbb{Z}_p$
  $c \leftarrow g^y$
  $K \leftarrow H(pk, c, X^y)$
  return $(K, c)$

Alg. Dec($par, sk, c$)
  $K \leftarrow H(pk, c, c^x)$
  return $K$
```

Fig. 5. Variants of Hashed-ElGamal KEM $\mathsf{HEG}[\mathsf{GGen}, \mathtt{high}]$, $\mathsf{HEG}[\mathsf{GGen}, \mathtt{med}]$, and $\mathsf{HEG}[\mathsf{GGen}, \mathtt{low}]$ 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.

$\mathsf{HEG}[\mathsf{GGen}, \mathtt{gran}]$ *is* (m, n)-CCA *secure. Formally, for every adversary* $\mathcal{A}$ *against game* $\mathrm{G}^{(m,n)\text{-cca}}_{\mathsf{HEG}[\mathsf{GGen},\mathsf{gran}]}$ *making at most* q *queries to random oracle* RO *there exists an adversary* $\mathcal{B}$ *against game* $\mathrm{G}^{(m,n)\text{-gcdh}}_{\mathsf{GGen},\mathsf{gran}}$ *that makes at most* q *queries to* DDH *and runs in essentially the same time as* $\mathcal{A}$ *and satisfies*

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

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

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

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,

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

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

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

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 Sects. 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 $\mathrm{MinTime}^{(m,n)\text{-gcdh}}_{\mathbb{F}_\ell^*,\texttt{gran}} = \mathrm{MinTime}^{(m,n)\text{-dl}}_{\mathbb{F}_\ell^*,\texttt{gran}}$ is given by

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

We obtain the scaling factor by dividing by $\mathrm{MinTime}^{(1,1)\text{-dl}}_{\mathbb{F}_\ell^*,\texttt{gran}} = L_\ell(1/3, 1.902)$. 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 $\delta \le 0.67$ we get $L_\ell(1/3, 1.902) \ge L_\ell(1/3, \delta + 1.232)$. 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 $\texttt{gran} \in \{\texttt{med}, \texttt{high}\}$ we obtain

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

Further, we get $L_\ell(1/3, \delta + 1.902) \le L_\ell(1/3, 2.007)$ for $\delta \le 0.105$. 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 $0.105 \le \delta \le 0.368$ 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:

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

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 time $O(\sqrt{p})$.

[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 $\Theta(\sqrt{m})$.

The lower bounds derived in Sect. 6 (Corollaries 2 and 3 and Theorem 5) 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 $p_1, \ldots, p_n$, where all p_i are roughly of size p.) 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 the full version of the paper [3] we additionally 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 Sect. 5.1 we define the PolyDL and SHS problems. In Sect. 5.2 we derive the bound on the security of GapDL in the high granularity setting, and further argue that it is optimal. Bounds for the cases of medium and low granularity can be found in the full version of the paper [3].

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 which 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 $1 \leq m \leq n$, $d \geq 1$, and consider game $\mathrm{G}^{(m,n)\text{-}d\text{-polydl}}_{\mathsf{GGen},\mathtt{gran}}(\mathcal{A})$ of Fig. 6 associated with 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 $\mathbb{Z}_{\boldsymbol{p}[i]}$.

```
Game $\mathrm{G}^{(m,n)\text{-}d\text{-polydl}}_{\mathsf{GGen},\mathsf{gran}}(\mathcal{A})$
$\boldsymbol{\mathcal{G}} \leftarrow_{\$} \mathsf{PGen}[\mathsf{gran}]$
$\boldsymbol{x}[\cdot] \leftarrow_{\$} \mathbb{Z}_{\boldsymbol{p}[\cdot]}$; $\boldsymbol{X}[\cdot] \leftarrow \boldsymbol{g}[\cdot]^{\boldsymbol{x}[\cdot]}$
$(L, \boldsymbol{x'}) \leftarrow_{\$} \mathcal{A}^{\mathrm{Eval}}(\boldsymbol{\mathcal{G}}, \boldsymbol{X})$
if $|L| < m$: return 0
if $\boldsymbol{x'}[L] = \boldsymbol{x}[L]$: return 1
else: return 0

Oracle $\mathrm{Eval}(i, f, \hat{\boldsymbol{X}})$
if $\deg f > d$: return 0
parse $\hat{\boldsymbol{X}}$ as $\boldsymbol{g}[i]^{\hat{\boldsymbol{x}}}$
if $\boldsymbol{g}[i]^{f(\hat{\boldsymbol{x}})} = 1$:
    return 1
else: return 0
```

Fig. 6. Security game $\mathrm{G}^{(m,n)\text{-}d\text{-polydl}}_{\mathsf{GGen},\mathsf{gran}}(\mathcal{A})$ relative to $\mathsf{GGen}, \mathsf{gran}$, 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 $|\hat{\boldsymbol{X}}|$ indeterminates. PGen is defined in Fig. 3.

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 $\hat{\boldsymbol{X}} = (\hat{\boldsymbol{X}}[1], \ldots, \hat{\boldsymbol{X}}[k])$, 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 $\hat{\boldsymbol{x}}$ of the input elements $\hat{\boldsymbol{X}}$ with respect to generator $\boldsymbol{g}[i]$, and then returns 1 if and only if $\boldsymbol{g}[i]^{f(\hat{\boldsymbol{x}}[1], \ldots, \hat{\boldsymbol{x}}[k])} = 1$.

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

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

The next definition extends the search-by-hyperplane-query problem (SHQ) by Yun [30].

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 $d, n \geq 1$, and consider game $\mathrm{G}^{n\text{-shs}_d}_p(\mathcal{A})$ of Fig. 7 associated with adversary $\mathcal{A}$. In the game, a vector $\boldsymbol{a}$ of length n is sampled, where $\boldsymbol{a}[i]$ is uniformly 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

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

```
Game $\mathrm{G}_p^{n\text{-shs}_d}(\mathcal{A})$
    $\boldsymbol{a} \leftarrow_{\$} \mathbb{Z}_p^n$
    $\boldsymbol{a'} \leftarrow_{\$} \mathcal{A}^{\mathrm{Eval}}(p)$
    if $\boldsymbol{a'} = \boldsymbol{a}$: return 1
    else: return 0

Oracle Eval($f$)
    if $\deg(f) > d$: return 0
    if $f(\boldsymbol{a}) = 0$: return 1
    else: return 0
```

Fig. 7. Security game $\mathrm{G}_p^{n\text{-shs}_d}(\mathcal{A})$ 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 $\mathbb{Z}_p[X_1, \ldots, X_n]$.

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 $\mathsf{GGen}_{\mathrm{gg}}$ 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$[\mathsf{GGen}_{\mathrm{gg}}, \mathtt{high}]$ that makes at most q queries to the group-operation oracle and q_{Eval} queries to oracle Eval:*

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

This extends [30, Corollary 2] from standard DL to the polycheck case. Most importantly, it allows us to prove the following corollary.

Corollary 1. *Let n be any positive integer and $\mathsf{GGen}_{\mathrm{gg}}$ 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$[\mathsf{GGen}_{\mathrm{gg}}, \mathtt{high}]$ that makes at most q queries to the group-operation oracle and q_{DDH} queries to the DDH oracle:*

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

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 $f(X_1, X_2, X_3) := X_1X_2 - X_3$. Then apply Theorem 3 with $d = 2$. □

The result is optimal. Concretely, in the full version of the paper [3] we construct an algorithm that solves (n, n)-GapDL$[\mathsf{GGen}_{\mathrm{gg}}, \mathtt{high}]$ in q group operations with success probability $(q^2/4np)^n$. Thus, for large p the fastest generic adversary solving (n, n)-GapDL$[\mathsf{GGen}_{\mathrm{gg}}, \mathtt{high}]$ with overwhelming success probability requires $\sqrt{np/e} \leq q \leq 2\sqrt{np}$ group operations.

The proof of Theorem 3 follows a structure similar to Yun [30]. 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 [30, Theorem 1] while additionally handling Eval queries. The proof can be found the full version of the paper [3].

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*

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

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.

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. The proof is in the full version of the paper [3].

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^{-n}. □

Remark 1. 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[$\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}$], for $m \lneq n$. 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 extends the one for standard multi-instance discrete logarithm [29, Section 3] by also simulating oracle Eval: It simply forwards every query to its own oracle.

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, Sect. 6.2 medium, and Sect. 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, in Theorem 4 we provide a generic 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 generic reduction from (m, n)-GapCDH to (m, m)-GapDL in the high-granularity setting is sketched below. The full proof can be found in the full version of the paper [3].

Theorem 4. *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* *there exists a generic reduction that constructs from any algebraic adversary $\mathcal{A}$ against game $\mathrm{G}^{(m,n)\text{-gcdh}}_{\mathsf{GGen},\mathtt{high}}$ an algebraic adversary $\mathcal{B}$ against $\mathrm{G}^{(m,m)\text{-gdl}}_{\mathsf{GGen},\mathtt{high}}$ 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 Sect. 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. Its proof can be found in the full version of the paper [3].

Corollary 2. *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:*

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

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 q group operations with success probability $(q^2/4mp)^m$ by using the generic adversary against high-granularity DL provided in the full version [3]. Thus, 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.

Proof Idea of Theorem 4. 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 $\operatorname{diag}(\boldsymbol{y})\boldsymbol{x} = A\boldsymbol{x} + B\boldsymbol{y} + \boldsymbol{c}$, where $\operatorname{diag}(\boldsymbol{y})$ is the diagonal matrix with the elements of $\boldsymbol{y}$ on the diagonal. Rearranging, we obtain

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

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

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

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

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

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.[5] 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 $\operatorname{diag}(\boldsymbol{y})\boldsymbol{x} = A\boldsymbol{x} + B\boldsymbol{y} + \boldsymbol{c}$, 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 $\boldsymbol{y}[i]$

[5] This expansion technique is originally from the work of Ying and Kunihiro [29].

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. □

Remark 2. Note that using Corollary 2 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 4. One could define a "polycheck CDH" problem in the same fashion as it is done for discrete logarithm in Sect. 5 (in short, (m, n)-d-PolyCDH). It is then immediate to extend Theorem 4 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, with an additional multiplicative factor of $(d/2)^m$ the advantage of any adversary against game (m, n)-d-PolyCDH[$\mathsf{GGen}_{\mathsf{gg}}, \mathtt{high}$] can be bounded as in Corollary 2.

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. The bound follows from observing that we can simulate the medium-granularity game starting from the high-granularity one. Then, we can apply Corollary 2 after counting the additional group queries by the simulation. For more details, we refer to the full version of the paper [3].

Corollary 3. *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*:*

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

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. 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 can be found in the full version of the paper [3].

Theorem 5. *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:*

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

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, each of the first m instances can be solved in time q/m with success probability $(q/m)^2/4p$ using the algorithm provided in the full version of the paper [3]. Thus, (m, n)-GapCDH$[\mathsf{GGen}_{\mathsf{gg}}, \mathtt{low}]$ can be solved in time q by independently running the single-instance adversary on the first m instances which results in a success probability of $(q^2/4m^2p)^m$. Further, 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. Benedikt Auerbach was supported by the European Research Council, ERC consolidator grant (682815-TOCNeT), and conducted part of this work at Ruhr University Bochum, supported by the ERC Project ERCC (FP7/615074) and the NRW Research Training Group SecHuman. Federico Giacon conducted part of this work at Ruhr University Bochum, supported by the ERC Project ERCC (FP7/615074). Eike Kiltz was supported by the ERC Project ERCC (FP7/615074), 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 (2001). https://doi.org/10.1007/3-540-45353-9_12
2. Adrian, D., et al.: 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, October 2015

3. Auerbach, B., Giacon, F., Kiltz, E.: Everybody's a target: scalability in public-key encryption. Cryptology ePrint Archive, Report 2019/364 (2019). https://eprint.iacr.org/2019/364
4. Barbulescu, R.: Algorithms for discrete logarithm in finite fields. Ph.D. thesis, University of Lorraine, Nancy, France (2013)
5. Barbulescu, R., Pierrot, C.: The multiple number field sieve for medium- and high-characteristic finite fields. LMS J. Computa. Math. **17**(A), 230–246 (2014)
6. Bartusek, J., Ma, F., Zhandry, M.: The distinction between fixed and random generators in group-based assumptions. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part II. LNCS, vol. 11693, pp. 801–830. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26951-7_27
7. 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 (2000). https://doi.org/10.1007/3-540-45539-6_18
8. 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 (2012). https://doi.org/10.1007/978-3-642-32009-5_19
9. Bellare, M., Rogaway, P.: Optimal asymmetric encryption. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 92–111. Springer, Heidelberg (1995). https://doi.org/10.1007/BFb0053428
10. 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 (2006). https://doi.org/10.1007/11761679_25
11. Bernstein, D.J., Lange, T.: Batch NFS. In: Joux, A., Youssef, A. (eds.) SAC 2014. LNCS, vol. 8781, pp. 38–58. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13051-4_3
12. Boneh, D., Boyen, X., Goh, E.-J.: Hierarchical identity based encryption with constant size ciphertext. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 440–456. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_26
13. 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, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_23
14. 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, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_14
15. Fouque, P.-A., Joux, A., Mavromati, C.: Multi-user collisions: applications to discrete logarithm, even-Mansour and PRINCE. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part I. LNCS, vol. 8873, pp. 420–438. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45611-8_22
16. 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, Cham (2018). https://doi.org/10.1007/978-3-319-96881-0_2
17. Garay, J.A., Johnson, D.S., Kiayias, A., Yung, M.: Resource-based corruptions and the combinatorics of hidden diversity. In: Kleinberg, R.D. (ed.) ITCS 2013, pp. 415–428. ACM, January 2013

18. Guillevic, A., Morain, F.: Discrete logarithms. In: Mrabet, N.E., Joye, M. (eds.) Guide to pairing-based cryptography. CRC Press/Taylor and Francis Group, December 2016
19. 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)
20. 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. Int. J.Inf. Secur. **3**(2), 86–98 (2004). https://doi.org/10.1007/s10207-004-0045-9
21. Hofheinz, D., Nguyen, N.K.: On tightly secure primitives in the multi-instance setting. In: Lin, D., Sako, K. (eds.) PKC 2019, Part I. LNCS, vol. 11442, pp. 581–611. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17253-4_20
22. 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 (2001). https://doi.org/10.1007/3-540-45537-X_17
23. Maurer, U.: Abstract models of computation in cryptography. In: Smart, N.P. (ed.) Cryptography and Coding 2005. LNCS, vol. 3796, pp. 1–12. Springer, Heidelberg (2005). https://doi.org/10.1007/11586821_1
24. Mizuide, T., Takayasu, A., Takagi, T.: Tight reductions for Diffie-Hellman variants in the algebraic group model. In: Matsui, M. (ed.) CT-RSA 2019. LNCS, vol. 11405, pp. 169–188. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-12612-4_9
25. Okamoto, T., Pointcheval, D.: The gap-problems: a new class of problems for the security of cryptographic schemes. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 104–118. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44586-2_8
26. Rupp, A., Leander, G., Bangerter, E., Dent, A.W., Sadeghi, A.-R.: Sufficient conditions for intractability over black-box groups: generic lower bounds for generalized DL and DH Problems. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 489–505. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89255-7_30
27. Sadeghi, A.-R., Steiner, M.: Assumptions related to discrete logarithms: why subtleties make a real difference. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 244–261. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44987-6_16
28. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 256–266. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-69053-0_18
29. 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 2017. LNCS, vol. 10355, pp. 498–517. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61204-1_25
30. 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 (2015). https://doi.org/10.1007/978-3-662-46803-6_27