A preliminary version of this paper appears in the proceedings of PKC 2025. This is the full version.

# Intermundium-DL: Assessing the Resilience of Current Schemes to Discrete-Log-Computation Attacks on Public Parameters

Mihir Bellare[1] Doreen Riepel[2] Laura Shea[3]

April 11, 2025

## Abstract

We consider adversaries able to perform a nonzero but small number of discrete logarithm computations, as would be expected with near-term quantum computers. Schemes with public parameters consisting of a few group elements are now at risk; could an adversary knowing the discrete logarithms of these elements go on to easily compromise the security of many users? We study this question for known schemes and find, across them, a perhaps surprising variance in the answers. In a first class are schemes, including Okamoto and Katz-Wang signatures, that we show fully retain security even when the discrete logs of the group elements in their parameters are known to the adversary. In a second class are schemes like Cramer-Shoup encryption and the SPAKE2 password-authenticated key exchange protocol that we show retain some partial but still meaningful and valuable security. In the last class are schemes we show by attack to totally break. The distinctions uncovered by these results shed light on the security of classical schemes in a setting of immediate importance, and help make choices moving forward.

[1] Department of Computer Science & Engineering, University of California San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. Email: `mbellare@ucsd.edu`. URL: `http://cseweb.ucsd.edu/~mihir/`. Supported in part by NSF grant CNS-2154272 and KACST.

[2] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany. Email: `riepel@cispa.de`. Work done while at UCSD, supported in part by KACST.

[3] Department of Computer Science & Engineering, University of California San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. Email: `lmshea@ucsd.edu`. Supported by NSF grants CNS-2048563, CNS-1513671 and CNS-2154272.

## Contents

## 1 Introduction

We suggest that in a relatively near term, we will see quantum computers with which computation of discrete logarithms in currently standardized groups is feasible, but still slow and costly. An attacker may be able to compute a few discrete logarithms, but not many. How might they best exploit this capability?

A natural and potentially lucrative target is public parameters, which, for many schemes in public-key cryptography, consist of a short list $\pi = (h_1, \ldots, h_w)$ of elements in an underlying group $\mathbb{G}$. Assumed to have been generated in a trusted way, the parameters are the same across some large number $n$ of users, each of whom individually generates her public and secret key as a function of them. Importantly, however, the schemes, and their current proofs of security, assume that the discrete logarithms $\log_g(h_1), \ldots, \log_g(h_w)$ are hidden and not available to the adversary. (Here $g$ is a fixed, public generator of $\mathbb{G}$.) Our adversary, however, can violate this assumption via an intensive computation that yields it $\log_g(h_1), \ldots, \log_g(h_w)$. What it hopes for is to be able to easily and quickly compromise the security of many users, maybe even all $n$ of them.

The question we ask in this paper is whether, for current schemes, such an adversary would succeed. Put another way, we want to assess the security of schemes when the attacker knows the discrete logarithms of the group elements in the parameters. The known proofs break down, but this does not necessarily mean attacks exist. In fact we show an interesting variation in security across schemes in the literature: some break fully, some only partially, and some (surprisingly) not at all, meaning security is provably retained. The message is that some schemes offer more resistance than others to discrete-log-computing attacks on parameters, and this should be a consideration in what we call Intermundium-DL, the era in which well-equipped attackers can compute a few, but just a few, discrete logarithms.

Computation is not the only way an attacker may know $\log_g(h_1), \ldots, \log_g(h_w)$. Through another lens, largely inspired by the Dual EC backdoor [22], the attacker could have picked, and retained as a backdoor, a list $(\beta_1, \ldots, \beta_w)$ of exponents, defined $h_i = g^{\beta_i}$ $(1 \leq i \leq w)$, and then used its political influence to have $(h_1, \ldots, h_w)$ installed as parameters. Our model and setting captures this as well, so that our results can also be seen as showing variations, across different schemes, in their resistance to backdooring of this form.

Below we discuss our setup, results and motivation in more detail. We then discuss related work on parameter-focused attacks.

Schemes with group-element parameters. Let $\mathbb{G}$ be a group of order $p$ with generator $g$. We consider cryptographic schemes whose public parameters $\pi = (h_1, \ldots, h_w)$ consist of a list of $w$ group elements $h_1, \ldots, h_w \in \mathbb{G}$. We refer to $w$ as the width. The scheme-prescribed parameter-generation algorithm $\mathsf{Pg}$ picks $x_1, \ldots, x_w \leftarrow_{\$} \mathbb{Z}_p$ and sets $h_i \leftarrow g^{x_i}$ $(1 \leq i \leq w)$. A user can now generate her public and secret keys as $(pk, sk) \leftarrow_{\$} \mathsf{Kg}(\pi)$ where $\mathsf{Kg}$ is a scheme-prescribed key-generation algorithm that depends, through its input, on $\pi$. We imagine a large number $n$ of users, all using the same $\pi$. We refer to such schemes as group-element parameter (GEP) schemes.

Importantly, the discrete logs $x_1, \ldots, x_w$ are not made public and are assumed unknown to the adversary. This assumption is made in the known proofs of security of all GEP schemes that we know or consider. In particular, known proofs offer no security guarantees if the $x_i$ values are revealed to the adversary.

There are many examples of such GEP schemes. Classical ones are the Okamoto signature scheme [49] and the Cramer-Shoup encryption scheme [25], both of which are GEP schemes with width $w = 1$. The Katz-Wang signature scheme [41] is also a GEP scheme with width $w = 1$. SPAKE2 [7,51], a PAKE protocol that is a draft standard [43], is a GEP scheme with width $w = 2$.

The KOY PAKE protocol [40] is a GEP scheme with width $w = 4$.

Security against dl-equipped adversaries. Let us refer to an adversary as dl-equipped if it knows the discrete logarithms $\log_g(h_1), \ldots, \log_g(h_w)$ of the group elements in the public parameters. In this paper, we aim to assess the security of GEP schemes against such adversaries. Why do this? To motivate the question, we consider three possible worlds in the vein of Impagliazzo [35].

**Opti-DL.** In the optimistic world, DL computation remains infeasible. On the one hand, in the space of classical computers, the currently known DL algorithms remain the best possible. On the other hand, quantum computers — which as we know can theoretically compute discrete logs in polynomial time [56] — encounter engineering bottlenecks due to which they are never built. In this world, our question is uninteresting, since the discrete logs of the parameters are safe. However, we do not believe we live in this world. The expectation is that quantum computers will advance, albeit slowly.

**Pessi-DL.** In the pessimistic world, blindingly fast quantum computers are right around the corner, and all DL-based cryptography is forfeit. In this case, DL-based schemes should be abandoned and our question is again uninteresting. However, we do not believe we live in this world, for although significant efforts are underway to build quantum computers, the progress is expensive and slow.

**Intermundium-DL.** In this place in between the above two worlds, discrete log computations are feasible but at a significant per-computation cost. This world may arise due to increasingly efficient but still slow quantum computers, or to marginally improved classical algorithms or implementations. The cost is large enough to preclude large-scale (per-user) DL computations, but may allow a well-equipped adversary to compute the $w$ discrete logs $x_1, \ldots, x_w$ underlying the parameters. This, we believe, *is* the world we live in, at least for the foreseeable future. The question of security of the scheme against dl-equipped adversaries now becomes very relevant.

Definitions with advice. Before we can meaningfully assess security against dl-equipped adversaries, we need to say what it means. Our requirement is simple but strong, namely that the usual security for the primitive continues to be provided even against a dl-equipped adversary. For example, for a signature scheme $\mathsf{S}$, the regular notion of security is UF-CMA in the $n$-user setting. Game $\mathbf{G}^{\text{uf-cma}}_{\mathsf{S},n}$ starts, in procedure Init, by generating and giving the adversary the parameters $\pi = (h_1, \ldots, h_w)$. We denote our notion A-UF-CMA. The game $\mathbf{G}^{\text{a-uf-cma}}_{\mathsf{S},n}$ capturing it is the same as $\mathbf{G}^{\text{uf-cma}}_{\mathsf{S},n}$ except that Init additionally gives the adversary $\log_g(h_1), \ldots, \log_g(h_w)$. We call this additional information *advice*, and it is the reason for the "A-" prefix in "A-UF-CMA." Other oracles (here, for signing) and the winning condition are the same. In particular, the adversary wins if it forges a signature for even one user. In this regard, our definition is strong; the concern we discussed above is that the adversary breaks security of all $n$ users, but our definition asks that it not break even one user. We similarly extend definitions for public-key encryption (obtaining A-IND-CPA, A-IND-CCA1, A-IND-CCA2) and Password-Authenticated Key Exchange (obtaining A-PAKE). We note that the definitions we give in the body are actually more general than discussed here, allowing alternative and arbitrary advice choices.

Security tiers. The fact that current proofs of security for GEP schemes deny the adversary $\log_g(h_1), \ldots, \log_g(h_w)$ may lead us to think this is necessary for security, meaning a dl-equipped adversary would be able to violate security. Somewhat surprisingly, we show that this is not always true, and that different schemes offer quite different types of resilience to such adversaries. We distinguish three tiers as follows. The information below is summarized in Figure 1.

**Fully A-secure.** These are schemes for which we can prove that regular security is retained even if adversaries are dl-equipped. In particular we show A-UF-CMA security for both Okamoto [49]

| Tier | Scheme | Result |
|---|---|---|
| Fully A-secure | Okamoto signatures<br>Katz-Wang signatures | A-UF-CMA security<br>under DL |
| Partially A-secure | SPAKE2 | A-PAKE security for good passwords,<br>under SCDH |
| | Cramer-Shoup encryption | A-IND-CCA1 security under DT-DDH |
| A-Insecure | KOY-style PAKE protocols | Attacks |

Figure 1: **Overview of results.** Schemes ranked from greatest to least in terms of A-security, meaning resistance to adversaries that know the discrete logarithms of the group elements in the public parameters.

and Katz-Wang [41] signatures, meaning these schemes remain UF-CMA-secure even if the discrete logs of the group elements in the parameters are known to the adversary. Such schemes are effectively "better than advertised," seeing no security degradation in Intermundium-DL.

**Partially A-secure.** These are schemes that, in the face of dl-equipped adversaries, may not provide the same security as before, but do retain and provide some meaningful and valuable security. Our first example is the IND-CCA2 Cramer-Shoup encryption scheme [25]. It is a GEP scheme with width 1. With regard to A-IND-CCA2, we have neither a proof nor an attack, but we are able to show A-IND-CCA1. We note that what in our framework is A-IND-CPA was shown by Rosulek in a StackExchange post [52]. (For completeness, we have stated this result in our paper.) Our second example is the draft-standard SPAKE2 protocol [7, 43, 51] for password-authenticated key exchange. We show that in the presence of dl-equipped adversaries, it will still provide security for users whose passwords are well chosen. This is significant because many, even if not all users, do take care to pick good passwords, and other PAKE protocols (see below) do not provide even this security.

**A-Insecure.** These are schemes that break totally in the face of dl-equipped adversaries, meaning knowledge of the discrete logs of the group elements in the parameters allows an attacker to easily compromise security of all users. This is true for the KOY PAKE protocol [40], as we show via an attack given in Section 5.2. We note that similar attacks work to break follow-up protocols including ones in the framework of Gennaro-Lindell [31], the protocol proposed by Jiang and Gong [36] which simplifies the design of KOY, and its generalization by Groce and Katz [34].

TECHNICAL OVERVIEW. We expand here on some of the computational assumptions and techniques underlying our results. The body of the paper gives formal definitions of security and of the assumptions, scheme descriptions, theorem statements and proofs.

Recall that Okamoto is a width-1 GEP scheme, so the public parameters consist of one group element $h_1$. The classical UF-CMA security of the scheme is shown under the standard DL assumption [49]. Given a UF-CMA adversary, the proof constructs an adversary that finds $\log_g(h_1)$. Since our dl-equipped adversary is given $\log_g(h_1)$ upfront, the same proof will of course not work to show A-UF-CMA security. Our approach is different. We prove A-UF-CMA security of Okamoto (Theorem 3.1) assuming regular UF-CMA security of the Schnorr signature scheme [55]. That is, given an A-UF-CMA adversary against Okamoto, we construct a UF-CMA adversary against Schnorr. Since the latter scheme is well known to be UF-CMA under the DL assumption [1, 55] and also (with tighter proofs) under other assumptions [15, 53], we conclude that Okamoto is A-UF-CMA-secure under the same assumptions and (since our reduction is tight) with the same tightness. In conclusion, resistance to dl-equipped adversaries is not only present but is proven without extra assumptions. We use the same approach for the Katz-Wang signature scheme [41], again proving its A-UF-CMA security (Theorem 3.2) assuming UF-CMA security of Schnorr and thus concluding A-UF-CMA security under DL and other assumptions.

The Cramer-Shoup public-key encryption scheme [25] is a width-1 GEP scheme, so again the public parameters consist of one group element $h_1$. The proof of IND-CCA2 is under DDH [25]. We prove A-IND-CCA1 (Theorem 4.1) under the delayed-target DDH (DT-DDH) assumption. The assumption was introduced by Lipmaa [44] (under a different name). We call it DT-DDH in light of [42] and of Freeman [29], who introduced DT-CDH. With regard to A-IND-CCA2 security of Cramer-Shoup, we leave as an open problem to either give an attack or give a proof under a reasonable assumption.

Moving on to PAKEs, we give an attack showing that SPAKE2 does not retain security against offline dictionary attacks in the face of a dl-equipped adversary. We then show (Theorem 5.3) that it does retain A-PAKE security against dl-equipped adversaries for users whose passwords have enough entropy. This proof is under the strong CDH (SCDH) assumption introduced in [4], and the result stands in contrast to the attacks that emerge for KOY and other PAKE protocols.

Converses of signature results. We have shown A-UF-CMA security of Okamoto and Katz-Wang (KW) assuming UF-CMA security of Schnorr. A natural question is whether this assumption is necessary. We show that it is, by establishing, in Appendix A, the converses of our signature results. Namely we show that A-UF-CMA of Okamoto implies UF-CMA of Schnorr and similarly A-UF-CMA of KW implies UF-CMA of Schnorr. This means A-UF-CMA of Okamoto, A-UF-CMA of KW and UF-CMA of Schnorr are all equivalent.

Discussion and related work. One way to avoid attacks on public parameters is to not have public parameters at all. There are a few ways to do this. First, for GEP schemes, each user could make her own independent choice of $\pi$ and put it in her public key. But this increases the size of public keys, which becomes a significant cost. Reduction in key size is one of the motivations for schemes having public parameters. Second, one could use schemes that are already parameter-free. Many such are known, and more could be invented. This works if one is designing a new system and has the freedom to pick the scheme. But GEP schemes are, or may, already be in use in legacy systems, or, like SPAKE2, be draft standards. The systems using them are not easily changed or replaced, so we need to understand the security of current GEP schemes in the face of dl-equipped adversaries. Our work has accordingly focused on existing schemes; we do not give new schemes as, for example, [9] do.

Our question is related to the quantum annoyance property [28] introduced in the context of password-authenticated key exchange (PAKE). Typically, a PAKE protocol should protect against offline dictionary attacks. If a quantum attacker additionally needs to solve one discrete log for each (online) password guess, we say that the protocol is also quantum annoying. In a sense, if public parameters allow one to break all protocol instances, it cannot be quantum annoying. One can view our results as an extension of this basic idea to other primitives.

BRT [18] introduced multi-instance security, which was further studied for public-key encryption by AGK [10]. The question here is how adversary effort scales with the number of users it can compromise. Viewing our results in this model, we could make statements of the form: GEP schemes that can be broken efficiently by a dl-equipped adversary do not offer any scalability. However, we are not claiming the converse, namely that GEP schemes secure against dl-equipped adversaries would be considered scalable under the definitions of [10, 18]. Meeting their definitions would involve, in our setting, considering all attacks that could be mounted using a small number of DL computations. But we consider only a single, natural attack, namely to compute the discrete logarithms of the group elements in the public parameters. Our model and results are in this way pragmatic.

The main motivating setting for Intermundium-DL is elliptic curve groups and quantum discrete-log-computing algorithms in them. For groups of integers modulo a prime, the situation is more delicate. Work including [8] shows that, in these groups, the NFS [24] can be implemented so that, after an expensive precomputation phase that depends only on the prime, individual discrete logs can, in an online phase, be computed more efficiently (although still with subexponential time). The Intermundium-DL setting is applicable here to the extent that the cost of the online phase remains large enough to preclude bulk discrete log computations. This may not always be true (with 512-bit primes, the online phase is reported in [8] as taking only minutes per discrete log) but could be true for large primes. For quantum adversaries however, and for elliptic curve groups, feasible precomputation that makes subsequent discrete log computations easy is, to the best of our knowledge, not known to be possible.

The backdooring perspective. Backdoors in, or subversion of, public parameters, has been a historical theme in cryptography. In our context it would mean that the entity generating $\pi = (h_1, \ldots, h_w)$ is untrusted. It may have itself picked exponents $\beta_1, \ldots, \beta_w$, set $h_i = g^{\beta_i}$ $(1 \leq i \leq w)$ and arranged for $\pi = (h_1, \ldots, h_w)$ to be published, while retaining the backdoor $(\beta_1, \ldots, \beta_w)$. But such an entity, knowing the discrete logarithms $\log_g(h_i)$ $(1 \leq i \leq w)$ is, in our language, just another dl-equipped adversary, and thus our results, both positive and negative, continue to hold. The positive ones in particular indicate that the schemes in question resist backdooring of this form. In our framework, Dual EC [22] emerges as a width-1 GEP scheme that is insecure against dl-equipped adversaries, again making a connection with classical backdooring work.

We note that backdooring can be prevented by requiring that $h_i = H(i)$ for some public hash function $H$. (The SPAKE2 draft standard [43] does precisely this.) But this is not applicable to all schemes, and one must still consider legacy systems where this step was not taken. Also, hash-based derivation of parameters will not change the picture for Intermundium-DL because the hashing does not prevent the adversary from computing the discrete logs of $h_1, \ldots, h_w$.

We do not claim to give results about all possible malicious parameter generation. We are rather concerned with one, natural form, where the adversary picks, and thus knows, the discrete logs of the public parameters. This captures Dual-EC-style backdoors but not necessarily all types of backdoors. Our formal definitions and results further assume that the discrete logs are randomly chosen, but can, in this regard be extended. In particular our theorems hold even if the adversary is allowed to select exponents $(\beta_1, \ldots, \beta_w)$ arbitrarily before publishing parameters $\pi = (g^{\beta_1}, \ldots, g^{\beta_w})$.

The takeaway is that the Intermundium-DL setting does capture some, natural backdoors, including realistic ones that have been implemented in the real world.

## 2 Preliminaries

Notation and terminology. By $\varepsilon$ we denote the empty string. By $|Z|$ we denote the length of a string $Z$. We denote a vector $\mathbf{v}$ in bold. Its length is $|\mathbf{v}|$ and for $1 \leq i \leq |\mathbf{v}|$ we refer to entry $i$ of the vector by $\mathbf{v}[i]$. To make clear that something is a boolean expression, we may use the notation $b \leftarrow [\![ expr ]\!]$ to evaluate the boolean expression $expr$ and assign the result to $b$.

If $\mathcal{X}$ is a finite set, we let $x \leftarrow\$ \mathcal{X}$ denote picking an element of $\mathcal{X}$ uniformly at random and assigning it to $x$. If $A$ is an algorithm, we let $y \leftarrow A[\mathrm{O}_1, \ldots](x_1, \ldots; r)$ denote running $A$ on inputs $x_1, \ldots$ and coins $r$ with oracle access to $\mathrm{O}_1, \ldots$, and assigning the output to $y$. We let $y \leftarrow\$ A[\mathrm{O}_1, \ldots](x_1, \ldots)$ be the result of picking $r$ at random and computing $y \leftarrow A[\mathrm{O}_1, \ldots](x_1, \ldots; r)$.

```
Game $\mathbf{G}^{\mathrm{dl}}_{\mathbb{G},p,g}$

INIT:
  $x \leftarrow\$ \mathbb{Z}_p$ ; $X \leftarrow g^x$
  Return $X$

FIN($x'$):
  Return $(g^{x'} = X)$
```

```
Game $\mathbf{G}^{\mathrm{ddh}}_{\mathbb{G},p,g}$

INIT:
  $d \leftarrow\$ \{0,1\}$
  $z \leftarrow\$ \mathbb{Z}_p$ ; $t \leftarrow\$ \mathbb{Z}_p$
  $Z \leftarrow g^z$ ; $T \leftarrow g^t$
  If $(d = 1)$ then $S \leftarrow g^{zt}$
  If $(d = 0)$ then $S \leftarrow\$ \mathbb{G}$
  Return $(Z, T, S)$

FIN($d'$):
  Return $(d' = d)$
```

```
Game $\mathbf{G}^{\mathrm{cdh}}_{\mathbb{G},p,g}$

INIT:
  $z \leftarrow\$ \mathbb{Z}_p$ ; $t \leftarrow\$ \mathbb{Z}_p$
  $Z \leftarrow g^z$ ; $T \leftarrow g^t$
  Return $(Z, T)$

FIN($S'$):
  Return $(S' = g^{zt})$
```

Figure 2: Standard computational problems over a cyclic group specified by $(\mathbb{G}, p, g)$.

We let $\mathrm{OUT}(A[\mathrm{O}_1, \ldots](x_1, \ldots))$ denote the set of all possible outputs of $A$ when invoked with inputs $x_1, \ldots$ and oracles $\mathrm{O}_1, \ldots$.

Algorithms are randomized unless otherwise indicated. Running time is worst case, which for an algorithm with access to oracles means across all possible replies from the oracles. An adversary is an algorithm. We use $\perp$ (bot) as a special symbol to denote rejection, and it is assumed to not be in $\{0,1\}^*$. We may interchangeably refer to the boolean false and integer 0, or to the boolean true and integer 1.

For sets $D, R$ we let $\mathsf{FUNC}(D, R)$ denote the set of all functions $f : D \to R$.

Games. We use the code-based game-playing framework of BR [20]. A game G starts with an optional INIT procedure, followed by a non-negative number of additional procedures called oracles, and ends with a FIN procedure. Execution of adversary $A$ with game G begins by running INIT (if present) to produce input $\leftarrow\$$ INIT. This INIT call precedes all other oracle calls, including an RO (if present). $A$ is then given input and is run with query access to the game oracles. When $A$ terminates with some output, execution of game G ends by returning FIN(output). By $\Pr[\mathrm{G}(A)]$ we denote the probability that the execution of game G with adversary $A$ results in FIN(output) being the boolean true.

Different games may have procedures (oracles) with the same names. If we need to disambiguate, we may write G.O to refer to oracle O of game G. In games, integer variables, set variables, boolean variables and string variables are assumed initialized, respectively, to 0, the empty set $\emptyset$, the boolean false and $\perp$. Tables are initialized with all entries being $\perp$. Games may occasionally Require: some condition, which means that all adversaries must obey this condition. This is used to rule out trivial wins.

Groups. In all of our considered schemes, the public parameters are based on a cyclic group. A *group specification* is a triple $(\mathbb{G}, p, g)$ where $\mathbb{G}$ is a cyclic group of order $p$ and $g$ is a generator of $\mathbb{G}$. We let $\mathsf{Gens}(\mathbb{G}) \subseteq \mathbb{G}$ be the set of generators of $\mathbb{G}$. Note that $\mathsf{Gens}(\mathbb{G}) = \{ g^x : x \in \mathbb{Z}_p^* \}$ and hence "$x \leftarrow\$ \mathbb{Z}_p^*$ ; return $g^x$" returns a random generator of $\mathbb{G}$. If $h \in \mathsf{Gens}(\mathbb{G})$ and $g' \in \mathbb{G}$ then $\log_h(g')$ is the unique $x \in \mathbb{Z}_p$ such that $h^x = g'$.

In general we do not assume $p$ is prime. This means our results are true for (cyclic) groups of any order. They hold not only for prime-order elliptic curve groups but also for groups of integers modulo a prime, which is important because the latter groups are still in use and are also of historical interest.

```
GPg_w:
    For $i = 1, \ldots, w$ do
        $\beta_i \leftarrow\!\$\ \mathbb{Z}_p$ ; $h_i \leftarrow g^{\beta_i}$
    Return $(h_1, \ldots, h_w)$

GPg*_w:
    For $i = 1, \ldots, w$ do
        $\beta_i \leftarrow\!\$\ \mathbb{Z}_p^*$ ; $h_i \leftarrow g^{\beta_i}$
    Return $(h_1, \ldots, h_w)$

AGPg_w:
    For $i = 1, \ldots, w$ do
        $\beta_i \leftarrow\!\$\ \mathbb{Z}_p$ ; $h_i \leftarrow g^{\beta_i}$
    $ad \leftarrow (\beta_1, \ldots, \beta_w)$
    Return $((h_1, \ldots, h_w), ad)$

AGPg*_w:
    For $i = 1, \ldots, w$ do
        $\beta_i \leftarrow\!\$\ \mathbb{Z}_p^*$ ; $h_i \leftarrow g^{\beta_i}$
    $ad \leftarrow (\beta_1, \ldots, \beta_w)$
    Return $((h_1, \ldots, h_w), ad)$
```

Figure 3: Group-parameter generation, and group-parameter-with-advice generation algorithms, for a group specification $(\mathbb{G}, p, g)$. The integer $w$ is called the width.

We recall some standard computational problems over groups in Figure 2. The Discrete Log (DL) problem is captured by game $\mathbf{G}^{\mathrm{dl}}_{\mathbb{G},p,g}$ for which the advantage of an adversary $A$ is $\mathbf{Adv}^{\mathrm{dl}}_{\mathbb{G},p,g}(A) = \Pr[\mathbf{G}^{\mathrm{dl}}_{\mathbb{G},p,g}(A)]$. Decisional Diffie-Hellman (DDH) is captured by game $\mathbf{G}^{\mathrm{ddh}}_{\mathbb{G},p,g}$ for which the advantage of an adversary $A$ is $\mathbf{Adv}^{\mathrm{ddh}}_{\mathbb{G},p,g}(A) = 2 \cdot \Pr[\mathbf{G}^{\mathrm{ddh}}_{\mathbb{G},p,g}(A)] - 1$. Computational Diffie-Hellman (CDH) is captured by game $\mathbf{G}^{\mathrm{cdh}}_{\mathbb{G},p,g}$ for which the advantage of an adversary $A$ is $\mathbf{Adv}^{\mathrm{cdh}}_{\mathbb{G},p,g}(A) = \Pr[\mathbf{G}^{\mathrm{cdh}}_{\mathbb{G},p,g}(A)]$. We will see variants of these hardness assumptions in Sections 4 and 5.

PARAMETER- AND ADVICE-GENERATION ALGORITHMS. Let $(\mathbb{G}, p, g)$ be a group specification as above. This paper considers schemes whose parameters are a list $(h_1, \ldots, h_w)$ of group elements. We refer to integer $w$ as the width. The left panel of Figure 3 shows honest parameter-generation algorithms that schemes would use to generate these parameters. We refer to these as the group-parameter generation algorithms. There are two types. The first $(\mathsf{GPg}_w)$ returns random group elements, and the second $(\mathsf{GPg}^*_w)$ returns random generators, corresponding to these two kinds of parameters arising in schemes.

We consider attacks, on parameters of the form $(h_1, \ldots, h_w) \in \mathbb{G}^w$, that allow the adversary to learn $\log_g(h_i)$ for all (or for some) $i$. We call these latter discrete log values the *natural advice.* This advice may be obtained by an adversary in a few ways. One is to compute it, for example using a quantum computer. In the Intermundium-DL setting, this reflects a world in which quantum DL computation is feasible but expensive, and we suppose an adversary uses its limited resources to compute the DLs of the public parameters. The backdooring perspective provides another way in which the adversary may have the discrete logs. Namely, the adversary itself runs parameter generation, thus picking the $\beta_i$ and letting $h_i \leftarrow g^{\beta_i}$. Now it may have the political or institutional power to influence standardization, causing $(h_1, \ldots, h_w)$ to be installed as the public parameters for users, while it retains the backdoor (advice) $(\beta_1, \ldots, \beta_w)$. In either case, thus, an adversary obtains both the public parameters $\pi = (h_1, \ldots, h_w)$ and the natural advice $ad = (\beta_1, \ldots, \beta_w)$.

To formally capture adversaries with this natural advice, we define in the right panel of Figure 3 two group-parameter-with-advice generation algorithms corresponding to the honest ones in the left panel. These return not just the parameters, but also the (natural) advice that we have just discussed. Again, there are two types, depending on whether parameters are random group elements or random generators. In the backdooring setting, the adversary is viewed as having

```
Games $\mathbf{G}^{\text{uf-cma}}_{\mathsf{S},n}$, $\mathbf{G}^{\text{a-uf-cma}}_{\mathsf{S},\mathsf{APg},n}$

INIT:
$\mathsf{H} \leftarrow\!\!\$\ \mathsf{S.HS}$
$\pi \leftarrow\!\!\$\ \mathsf{S.Pg}[\mathsf{H}]$ // Game $\mathbf{G}^{\text{uf-cma}}_{\mathsf{S},n}$ only
$(\pi, ad) \leftarrow\!\!\$\ \mathsf{APg}[\mathsf{H}]$ // Game $\mathbf{G}^{\text{a-uf-cma}}_{\mathsf{S},\mathsf{APg},n}$ only
For $i = 1 \ldots n$ do $(\mathbf{vk}[i], \mathbf{sk}[i]) \leftarrow\!\!\$\ \mathsf{S.Kg}[\mathsf{H}](\pi)$
Return $(\pi, \mathbf{vk})$ // Game $\mathbf{G}^{\text{uf-cma}}_{\mathsf{S},n}$ only
Return $(\pi, \mathbf{vk}, ad)$ // Game $\mathbf{G}^{\text{a-uf-cma}}_{\mathsf{S},\mathsf{APg},n}$ only

SIGN$(i, m)$:
$\sigma \leftarrow\!\!\$\ \mathsf{S.Sign}[\mathsf{H}](\pi, \mathbf{sk}[i], m)$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(i, m)\}$ ; Return $\sigma$

HASH$(z)$:
Return $\mathsf{H}(z)$

FIN$(i, m, \sigma)$:
If $(i, m) \in \mathcal{Q}$ then return 0
Return $\mathsf{S.Vfy}[\mathsf{H}](\pi, \mathbf{vk}[i], m, \sigma)$
```

Figure 4: Games defining UF-CMA and A-UF-CMA of a signature scheme $\mathsf{S}$ over $n$ users, for a particular advice-generation algorithm $\mathsf{APg}$.

run these advice algorithms rather than the honest ones, and retained the advice (backdoor) $ad$. We clarify that in the Intermundium-DL setting, nobody actually runs these algorithms. They function, rather, as an abstraction to capture that the adversary knows $(\beta_1, \ldots, \beta_w)$ through a computation on $(h_1, \ldots, h_w)$.

Schemes we will consider will be prescribed as using $\mathsf{GPg}_w$ or $\mathsf{GPg}^*_w$ for some $w$ associated to the scheme. When we consider security under parameter subversion, we will replace these with $\mathsf{AGPg}_w$ or $\mathsf{AGPg}^*_w$, respectively. Specifically, we will see $\mathsf{AGPg}^*_1$ in Sections 3 and 4, and $\mathsf{AGPg}_2, \mathsf{AGPg}^*_4$ in Section 5.2.

## 3 Signatures under parameter subversion

In this section we provide positive results for existing signature schemes where the setup appears susceptible to parameter attacks, showing that these schemes retain security even if the adversary knows the discrete logs of the parameters. This shows that the natural attacks on parameters are not effective.

### 3.1 Signature definitions

A signature scheme $\mathsf{S}$ specifies an (honest) parameter-generation algorithm $\mathsf{S.Pg}$, a key-generation algorithm $\mathsf{S.Kg}$, a signing algorithm $\mathsf{S.Sign}$ and a deterministic verification function $\mathsf{S.Vfy}$. If the scheme requires a hash function mapping from some domain $D$ to range $R$, the scheme defines a set $\mathsf{S.HS} \subseteq \mathsf{FUNC}(D, R)$ from which the hash function $\mathsf{H} \leftarrow\!\!\$\ \mathsf{S.HS}$ is drawn. Making the set $\mathsf{S.HS}$ a scheme parameter yields flexibility. We can set $\mathsf{S.HS} = \mathsf{FUNC}(D, R)$ to capture the ROM. We can also set $\mathsf{S.HS}$ to the singleton set $\{\,\text{SHA256}\,\}$, for example, to capture the scheme using a particular hash function. We introduce this flexibility because our results in this section work with different

```
OS.Pg[H]:
    $\beta \leftarrow\!\$\, \mathbb{Z}_p^*$ ; $h \leftarrow g^\beta$
    Return $h$

OS.Kg[H]($h$):
    $s_1, s_2 \leftarrow\!\$\, \mathbb{Z}_p$
    $X \leftarrow g^{s_1} h^{s_2}$
    Return $(X, (s_1, s_2))$

OS.Sign[H]($h, (s_1, s_2), m$):
    $r_1, r_2 \leftarrow\!\$\, \mathbb{Z}_p$
    $R \leftarrow g^{r_1} h^{r_2}$
    $e \leftarrow \mathsf{H}(X, R, m)$
    $y_1 \leftarrow (r_1 + es_1) \bmod p$
    $y_2 \leftarrow (r_2 + es_2) \bmod p$
    Return $\sigma \leftarrow (e, y_1, y_2)$

OS.Vfy[H]($h, X, m, \sigma$):
    $(e, y_1, y_2) \leftarrow \sigma$
    $R \leftarrow g^{y_1} h^{y_2} X^{-e}$
    If $\mathsf{H}(X, R, m) = e$ then return 1
    Return 0
```

```
SS.Kg[H]:
    $s \leftarrow\!\$\, \mathbb{Z}_p$
    $X \leftarrow g^s$
    Return $(X, s)$

SS.Sign[H]($s, m$):
    $r \leftarrow\!\$\, \mathbb{Z}_p$
    $R \leftarrow g^r$
    $e \leftarrow \mathsf{H}(X, R, m)$
    $y \leftarrow (r + es) \bmod p$
    Return $\sigma \leftarrow (e, y)$

SS.Vfy[H]($X, m, \sigma$):
    $(e, y) \leftarrow \sigma$
    $R \leftarrow g^y X^{-e}$
    If $\mathsf{H}(X, R, m) = e$ then return 1
    Return 0
```

Figure 5: Okamoto signature scheme OS (left) and Schnorr signature scheme SS (right) associated to group specification $(\mathbb{G}, p, g)$ and set $\overline{\mathsf{HS}}$ of hash functions, with $\mathsf{H} \in \overline{\mathsf{HS}}$.

choices of this hash-function space, either requiring the ROM or holding for any standard-model choice of hash function.

We generate parameters as $\pi \leftarrow\!\$\, \mathsf{S.Pg}[\mathsf{H}]$ and keys (for verification and signing, respectively) via $(vk, sk) \leftarrow\!\$\, \mathsf{S.Kg}[\mathsf{H}](\pi)$. Signing takes as input $\pi$ along with a signing key $sk$ and message $m$ to return signature $\sigma \leftarrow\!\$\, \mathsf{S.Sign}[\mathsf{H}](\pi, sk, m)$. Verification outputs a bit $d$ via $d \leftarrow \mathsf{S.Vfy}[\mathsf{H}](\pi, vk, m, \sigma)$. For correctness we require that $\mathsf{S.Vfy}[\mathsf{H}](\pi, vk, m, \mathsf{S.Sign}[\mathsf{H}](\pi, sk, m)) = 1$ for all $\pi \in \mathrm{OUT}(\mathsf{S.Pg}[\mathsf{H}])$, all $(vk, sk) \in \mathrm{OUT}(\mathsf{S.Kg}[\mathsf{H}](\pi))$, all $m \in \{0,1\}^*$ and all $\mathsf{H} \in \mathsf{S.HS}$.

Game $\mathbf{G}^{\text{uf-cma}}_{\mathsf{S},n}$ in Figure 4 defines the usual UF-CMA security, where parameters $\pi$ are honestly generated and we are in the multi-user setting [14, 46] with $n > 0$ users. Our new notion of A-UF-CMA security is defined in game $\mathbf{G}^{\text{a-uf-cma}}_{\mathsf{S},\mathsf{APg},n}$ in the same figure. The game is additionally parameterized by an advice-generation algorithm APg that returns not just parameters $\pi$, but related advice $ad$, both of which are returned to the adversary. In this way, the game captures the adversary's ability to violate UF-CMA when, in addition to the parameters, it also has advice related to them. The advice-generation algorithm APg is not part of the scheme S, but is instead an external parameter. The games share many lines, the exceptions indicated in the comments. For an adversary $A$ we let $\mathbf{Adv}^{\text{uf-cma}}_{\mathsf{S},n}(A) = \Pr[\mathbf{G}^{\text{uf-cma}}_{\mathsf{S},n}(A)]$, and we let $\mathbf{Adv}^{\text{a-uf-cma}}_{\mathsf{S},\mathsf{APg},n}(A) = \Pr[\mathbf{G}^{\text{a-uf-cma}}_{\mathsf{S},\mathsf{APg},n}(A)]$.

The particular schemes in this section will have $\mathsf{S.Pg} = \mathsf{GPg}_1^*$ so we will correspondingly set $\mathsf{APg} = \mathsf{AGPg}_1^*$. (These algorithms are shown in Figure 3.)

## 3.2 Positive signature results: Okamoto

Okamoto and Schnorr schemes. Let $(\mathbb{G}, p, g)$ be a group specification. Let $\overline{\mathsf{HS}} \subseteq \mathsf{FUNC}(\mathbb{G} \times \mathbb{G} \times \{0,1\}^*, \mathbb{Z}_p)$ be a non-empty set of functions. (Functions from this set will serve as hash functions for the schemes we now define.) We associate to $(\mathbb{G}, p, g)$ and $\overline{\mathsf{HS}}$ the Okamoto signature scheme $\mathsf{OS}$ and also the Schnorr signature scheme $\mathsf{SS}$. Their algorithms are shown in Figure 5. Notice that the parameter-generation algorithm for Okamoto is $\mathsf{OS.Pg} = \mathsf{GPg}_1^*$, so that, when considering A-UF-CMA, we will use $\mathsf{AGPg}_1^*$ in its place. The Schnorr scheme does not use parameters, so we do not show a parameter-generation algorithm. We set $\mathsf{OS.HS} = \mathsf{SS.HS} = \overline{\mathsf{HS}}$, to indicate that both schemes use a hash function $\mathsf{H} : \mathbb{G} \times \mathbb{G} \times \{0,1\}^* \to \mathbb{Z}_p$ from the same set.

In the literature, these schemes come in a few variants corresponding to a few choices. For consistency, we have fixed some choices in the same way for both schemes. To elaborate, first, the schemes both include a hashing step of the form $\mathsf{H}(vk, R, m)$. The values $R, m$ are always included and $vk$ is sometimes included for tighter security, following [21, 30]. We have decided to always include $vk$, but our results hold without it. Second, in both schemes, in reference to the same hashing step, either $R$ or $e = \mathsf{H}(vk, R, m)$ is returned as a component of a signature. ($R$ can be recovered from $e$ and public information.) For Schnorr these two variants are proved equivalent [12]. (We are not aware of such a proof for Okamoto.) In this paper we consistently include $e$ in the signature. Third, while our Schnorr verification key is $X = g^s$, another option is $X = g^{-s}$, with a corresponding adjustment in verification. Likewise, the Okamoto verification key could be $g^{-s_1}h^{-s_2}$ instead. Our proofs work in both cases, but we warn that the change must be consistent across the two schemes, meaning, for both schemes, one either uses the current version or the option with negation in the exponent.

Security of Okamoto against parameter attacks. We now ask whether knowledge of the discrete logarithm $\beta = \log_g(h)$ allows an attacker to forge Okamoto signatures. It is worth noting that existing security proofs of the UF-CMA security of Okamoto [1, 49] rely on the hardness of finding the discrete logarithm of $h$. Specifically, these proofs take an adversary against Okamoto and build an adversary that finds $\beta = \log_g(h)$. This may seem to indicate that $\beta$ would allow an attacker to violate security of Okamoto. Maybe somewhat surprisingly, we show this is not true. We provide an alternative proof showing security of Okamoto even against adversaries that know $\beta$. Formally, in our framework, this is a proof of A-UF-CMA for $\mathsf{OS}$ relative to advice-generation algorithm $\mathsf{AGPg}_1^*$. The assumption we make for this proof is the (standard) UF-CMA security of the Schnorr signature scheme. (The latter is well understood and has been shown under many different assumptions and with different degrees of tightness [1, 15, 50, 53, 54].) Our result is stated below.

**Theorem 3.1** *Let $(\mathbb{G}, p, g)$ be a group specification. Let $\overline{\mathsf{HS}} \subseteq \mathsf{FUNC}(\mathbb{G} \times \mathbb{G} \times \{0,1\}^*, \mathbb{Z}_p)$ be a non-empty set of functions. Let $\mathsf{OS}$ and $\mathsf{SS}$ be the Okamoto and Schnorr signature schemes, respectively, associated to $(\mathbb{G}, p, g)$ and $\overline{\mathsf{HS}}$ as above. Let $n > 0$ be a number of users. Given an adversary $B$ in game $\mathbf{G}^{\text{a-uf-cma}}_{\mathsf{OS},\mathsf{AGPg}_1^*,n}$ we can construct an adversary $A$ in game $\mathbf{G}^{\text{uf-cma}}_{\mathsf{SS},n}$ such that*

$$\mathbf{Adv}^{\text{a-uf-cma}}_{\mathsf{OS},\mathsf{AGPg}_1^*,n}(B) \leq \mathbf{Adv}^{\text{uf-cma}}_{\mathsf{SS},n}(A) . \tag{1}$$

*Adversary $A$ makes the same number of signing and hashing queries as $B$, and has running time close to that of $B$.*

Before proving the theorem, we make a few remarks. First, the set $\overline{\mathsf{HS}}$ of hash functions is arbitrary. As per the discussion above, setting $\overline{\mathsf{HS}} = \mathsf{FUNC}(\mathbb{G} \times \mathbb{G} \times \{0,1\}^*, \mathbb{Z}_p)$ captures the ROM, so in particular the result holds there. But it also holds in the standard model, for example when the

```
Adversary $A(\varepsilon, \mathbf{X})$:
$\beta \leftarrow\!\!\$\ \mathbb{Z}_p^*$ ; $h \leftarrow g^\beta$
$(i, m, \sigma) \leftarrow B[\textsc{SignSim}, \text{HSim}](h, \mathbf{X}, \beta)$
$(e, y_1, y_2) \leftarrow \sigma$
$y \leftarrow (y_1 + \beta y_2) \bmod p$
Return $(i, m, (e, y))$

Subroutine $\textsc{SignSim}(i, m)$:
$\sigma \leftarrow \mathbf{G}^{\text{uf-cma}}_{\mathsf{SS},n}.\textsc{Sign}(i, m)$
$(e, y) \leftarrow \sigma$
$y_2 \leftarrow\!\!\$\ \mathbb{Z}_p$ ; $y_1 \leftarrow (y - \beta y_2) \bmod p$
$\sigma' \leftarrow (e, y_1, y_2)$
Return $\sigma'$

Subroutine $\text{HSim}(z)$:
Return $\mathbf{G}^{\text{uf-cma}}_{\mathsf{SS},n}.\textsc{Hash}(z)$
```

Figure 6: Adversary $A$ for the proof of Theorem 3.1.

schemes use one particular hash function like SHA256, captured by setting $\overline{\mathsf{HS}}$ to the singleton set consisting of just this one hash function. Second, the reduction of Eq. (1) is tight, meaning there is no tightness loss relative to Schnorr. Leveraging known results about the latter, this actually brings new results even on the classical UF-CMA security of Okamoto. Since A-UF-CMA implies UF-CMA, Theorem 3.1 says that UF-CMA security of Okamoto reduces tightly to that of Schnorr, which, beyond the usual results from DL, yields tighter proofs for Okamoto from the MBDL assumption via [15] and also from the 2-moment-DL assumption via [53].

One may ask whether an assumption weaker than UF-CMA of Schnorr suffices to prove A-UF-CMA of Okamoto. Towards this, we give in Appendix A a result that is the converse of Theorem 3.1. It shows that A-UF-CMA of Okamoto implies UF-CMA of Schnorr. This means that A-UF-CMA of Okamoto and UF-CMA of Schnorr are in fact equivalent.

**Proof of Theorem 3.1:** We give adversary $A$ in Figure 6. It is playing game $\mathbf{G}^{\text{uf-cma}}_{\mathsf{SS},n}$. Accordingly it receives as input empty public parameters (since $\pi = \varepsilon$ for the Schnorr scheme) and a list $\mathbf{X}$ of verification keys where $\mathbf{X}[i] = g^{\mathbf{s}[i]}$. It itself picks the advice $\beta$ to compute $h$ and runs adversary $B$ on input $(\pi, \mathbf{X}, ad)$ with $\pi = h$ and $ad = \beta$. It defines subroutine $\textsc{SignSim}$ to answer $B$'s signing queries; this calls $A$'s own signing oracle to compute the simulated signatures as shown. $A$ responds to $\textsc{Hash}$ queries of $B$ via its own $\textsc{Hash}$, through subroutine HSim. When $B$ returns its forgery at line 2, $A$ processes it as shown at lines 3,4 to return its own forgery at line 5. Now we have to show that this works. This involves a few claims. First, we have to show that simulated signatures are properly distributed, then we have to show that if $B$'s forgery is correct, so is that of $A$. The former is somewhat delicate, and our approach is to use a game sequence in which we slowly morph Okamoto signatures into Schnorr signatures.

Thus, consider the games of Figure 7, to be executed with adversary $B$. Game $\text{G}_0$ is simply game $\mathbf{G}^{\text{a-uf-cma}}_{\mathsf{OS},\mathsf{AGPg}_1^*,n}$, the A-UF-CMA game for the Okamoto scheme $\mathsf{OS}$, and hence by definition we have

$$\mathbf{Adv}^{\text{a-uf-cma}}_{\mathsf{OS},\mathsf{AGPg}_1^*,n}(B) = \Pr[\text{G}_0(B)] \ .$$

Next, consider game $\text{G}_1$, which includes the boxed code. It defines the verification key $\mathbf{X}[i]$ differently. Namely, it picks $\mathbf{s}[i], \mathbf{s}_2[i]$ at random at line 4, defines $\mathbf{s}_1[i]$ at line 5 and then sets $\mathbf{X}[i]$ as shown at line 6. We claim this is equivalent to the choice made by $\text{G}_0$. This is true because $\mathbf{s}[i] = (\mathbf{s}_1[i] + \beta\mathbf{s}_2[i]) \bmod p$ and $h = g^\beta$ so $g^{\mathbf{s}[i]} = g^{\mathbf{s}_1[i]+\beta\mathbf{s}_2[i]} = g^{\mathbf{s}_1[i]}h^{\mathbf{s}_2[i]}$. Furthermore, the distributions of $\mathbf{s}_1, \mathbf{s}_2$ are identical in the two games. Likewise, $\textsc{Sign}$ in $\text{G}_1$ picks signatures differently from, but in a way we argue is equivalent to, $\text{G}_0$. At line 8 of $\text{G}_1$, values $r, y_2$ are chosen at random, and $R$ is defined as $g^r$. Lines 10,11 now define $r_2, r_1$ in such a way that $y_2 = (r_2 + e\mathbf{s}_2[i]) \bmod p$

```
Game $G_0$

INIT:
1  $\mathsf{H} \leftarrow\!\!\$\ \overline{\mathsf{HS}}$
2  $\beta \leftarrow\!\!\$\ \mathbb{Z}_p^*$ ; $h \leftarrow g^\beta$
3  For $i = 1, \ldots, n$ do
4      $\mathbf{s}_1[i], \mathbf{s}_2[i] \leftarrow\!\!\$\ \mathbb{Z}_p$
5      $\mathbf{X}[i] \leftarrow g^{\mathbf{s}_1[i]} h^{\mathbf{s}_2[i]}$
6  Return $(h, \mathbf{X}, \beta)$

SIGN$(i, m)$:
7  $r_1, r_2 \leftarrow\!\!\$\ \mathbb{Z}_p$ ; $R \leftarrow g^{r_1} h^{r_2}$
8  $e \leftarrow \mathsf{H}(\mathbf{X}[i], R, m)$
9  $y_1 \leftarrow (r_1 + e\mathbf{s}_1[i]) \bmod p$
10 $y_2 \leftarrow (r_2 + e\mathbf{s}_2[i]) \bmod p$
11 $\sigma \leftarrow (e, y_1, y_2)$
12 $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(i, m)\}$ ; Return $\sigma$

HASH$(z)$:
13 Return $\mathsf{H}(z)$

FIN$(i, m, \sigma)$:
14 If $(i, m) \in \mathcal{Q}$ then return 0
15 $(e, y_1, y_2) \leftarrow \sigma$
16 $R \leftarrow g^{y_1} h^{y_2} \mathbf{X}[i]^{-e}$
17 If $\mathsf{H}(\mathbf{X}[i], R, m) = e$ then return 1
18 Return 0
```

```
Games $\boxed{G_1}$, $G_2$

INIT:
1  $\mathsf{H} \leftarrow\!\!\$\ \overline{\mathsf{HS}}$
2  $\beta \leftarrow\!\!\$\ \mathbb{Z}_p^*$ ; $h \leftarrow g^\beta$
3  For $i = 1, \ldots, n$ do
4      $\mathbf{s}[i] \leftarrow\!\!\$\ \mathbb{Z}_p$ ; $\boxed{\mathbf{s}_2[i] \leftarrow\!\!\$\ \mathbb{Z}_p}$
5      $\boxed{\mathbf{s}_1[i] \leftarrow (\mathbf{s}[i] - \beta\mathbf{s}_2[i]) \bmod p}$
6      $\mathbf{X}[i] \leftarrow g^{\mathbf{s}[i]}$
7  Return $(h, \mathbf{X}, \beta)$

SIGN$(i, m)$:
8  $r, y_2 \leftarrow\!\!\$\ \mathbb{Z}_p$ ; $R \leftarrow g^r$
9  $e \leftarrow \mathsf{H}(\mathbf{X}[i], R, m)$
10 $\boxed{r_2 \leftarrow (y_2 - e\mathbf{s}_2[i]) \bmod p}$
11 $\boxed{r_1 \leftarrow (r - \beta r_2) \bmod p}$
12 $y \leftarrow (r + e\mathbf{s}[i]) \bmod p$
13 $y_1 \leftarrow (y - \beta y_2) \bmod p$
14 $\sigma \leftarrow (e, y_1, y_2)$
15 $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(i, m)\}$ ; Return $\sigma$

HASH$(z)$:
16 Return $\mathsf{H}(z)$

FIN$(i, m, \sigma)$:
17 If $(i, m) \in \mathcal{Q}$ then return 0
18 $(e, y_1, y_2) \leftarrow \sigma$
19 $y \leftarrow (y_1 + \beta y_2) \bmod p$
20 $R \leftarrow g^y \mathbf{X}[i]^{-e}$
21 If $\mathsf{H}(\mathbf{X}[i], R, m) = e$ then return 1
22 Return 0
```

Figure 7: Games for proof of Theorem 3.1. $G_1$ includes the boxed code and $G_2$ does not.

and $r = (r_1 + \beta r_2) \bmod p$. Due to the latter and $h = g^\beta$ we have $R = g^r = g^{r_1} h^{r_2}$ and furthermore $r_1, r_2$ are uniformly and independently distributed over $\mathbb{Z}_p$, as in $G_0$. Line 12 defines a value $y$ so that at line 13 we have

$$\begin{aligned} y_1 = (y - \beta y_2) \bmod p &= (r + e\mathbf{s}[i] - \beta y_2) \bmod p \\ &= (r_1 + \beta r_2 + e(\mathbf{s}_1[i] + \beta \mathbf{s}_2[i]) - \beta y_2) \bmod p \\ &= (r_1 + \beta(y_2 - e\mathbf{s}_2[i]) + e(\mathbf{s}_1[i] + \beta \mathbf{s}_2[i]) - \beta y_2) \bmod p \\ &= (r_1 + e\mathbf{s}_1[i]) \bmod p \end{aligned}$$

as in $G_0$. So the distribution of returned signatures in $G_1$ matches that in $G_0$. Finally, game $G_1$ checks the forgery at lines 19,20 somewhat differently through first defining $y$, but this is equivalent to the check in $G_0$ because $h = g^\beta$. At this point we have shown that

$$\Pr[G_1(B)] = \Pr[G_0(B)] \; . \tag{2}$$

```
KW.Pg[H]:
    $\beta \leftarrow\!\$\ \mathbb{Z}_p^*$ ; $h \leftarrow g^\beta$
    Return $h$

KW.Kg[H]($h$):
    $s \leftarrow\!\$\ \mathbb{Z}_p$
    $X_g \leftarrow g^s$ ; $X_h \leftarrow h^s$
    Return $((X_g, X_h), s)$

KW.Sign[H]($h, s, m$):
    $r \leftarrow\!\$\ \mathbb{Z}_p$
    $R_g \leftarrow g^r$ ; $R_h \leftarrow h^r$
    $e \leftarrow \mathsf{H}((X_g, X_h), R_g, R_h, m)$
    $y \leftarrow (r + es) \bmod p$
    Return $\sigma \leftarrow (e, y)$

KW.Vfy[H]($h, (X_g, X_h), m, \sigma$):
    $(e, y) \leftarrow \sigma$
    $R_g \leftarrow g^y X_g^{-e}$ ; $R_h \leftarrow h^y X_h^{-e}$
    If $\mathsf{H}((X_g, X_h), R_g, R_h, m) = e$ then return 1
    Return 0
```

Figure 8: Katz-Wang signature scheme KW associated to group specification $(\mathbb{G}, p, g)$, with hash function $\mathsf{H} \in \mathsf{KW.HS}$.

Now observe that in game $G_1$, the values computed in the boxed code are never used in computing what an oracle returns to the adversary. (We only used these values to argue that Eq. (2) is true.) So game $G_2$ simply drops these lines. Clearly, since as we just said these lines do not affect oracle responses, we have

$$\Pr[G_2(B)] = \Pr[G_1(B)] .$$

We now observe that in $G_2$ the keys are Schnorr ones, and $(e, y)$ is a Schnorr signature, as a function of which the Okamoto signature is computed. This allows us to see that

$$\mathbf{Adv}^{\text{uf-cma}}_{\mathsf{SS},n}(A) \geq \Pr[G_2(B)] .$$

For this (which concludes the proof) we identify the $\mathbf{s}$ in game $G_2$ with the secret keys chosen in $\mathbf{G}^{\text{uf-cma}}_{\mathsf{SS},n}$. Then we see the Schnorr signature $(e, y)$ in $G_2$ as the one returned at line 6 of SignSim in Figure 6. The subsequent lines of SignSim now mimic $G_2$ in how the Okamoto signature $(e, y_1, y_2)$ is computed. Finally, the forgery at line 5 of Figure 6 mimics Fin of $G_2$. Adversary $A$ of Figure 6 makes the same number of signing and hashing queries as $B$, as claimed. ▌

## 3.3 Positive signature results: Katz-Wang

Katz-Wang signatures. We now turn to proving a similar positive result for the Katz-Wang signature scheme [41]. The scheme, associated to a group specification $(\mathbb{G}, p, g)$, is denoted KW. Its algorithms are shown in Figure 8. Note that the parameter-generation algorithm is again $\mathsf{KW.Pg} = \mathsf{GPg}_1^*$, so the advice-generation algorithm for A-UF-CMA will be $\mathsf{AGPg}_1^*$. The scheme uses a hash function $\mathsf{H} : (\mathbb{G} \times \mathbb{G}) \times \mathbb{G} \times \mathbb{G} \times \{0,1\}^* \to \mathbb{Z}_p$. We set $\mathsf{KW.HS} = \mathsf{FUNC}((\mathbb{G} \times \mathbb{G}) \times \mathbb{G} \times \mathbb{G} \times \{0,1\}^*, \mathbb{Z}_p)$ to the full space of all such functions, reflecting that here, unlike in Theorem 3.1, we will rely on, not just allow, the ROM.

Security of Katz-Wang against parameter attacks. The KW scheme is shown in [41] to be UF-CMA under the DDH assumption. Their proof assumes honest parameter generation, so that in particular the discrete log $\beta = \log_g(h)$ of the parameter $h$ is not given to the adversary, a fact that the proof crucially exploits. We ask what happens if, instead, the attacker does have $\beta$. That is, we ask about A-UF-CMA relative to advice-generation algorithm $\mathsf{AGPg}_1^*$. As with

```
Adversary $A(\varepsilon, \mathbf{X})$:
$\beta \leftarrow\$ \mathbb{Z}_p^*$ ; $h \leftarrow g^\beta$
For $i = 1 \ldots n$ do $\mathbf{X}_g[i] \leftarrow \mathbf{X}[i]$ ; $\mathbf{X}_h[i] \leftarrow \mathbf{X}[i]^\beta$ ; $\overline{\mathbf{X}}[i] \leftarrow (\mathbf{X}_g[i], \mathbf{X}_h[i])$
$(i, m, \sigma) \leftarrow B[\textsc{SignSim}, \textsc{HSim}](h, \overline{\mathbf{X}}, \beta)$
Return $(i, m, \sigma)$

Subroutine $\textsc{SignSim}(i, m)$:
$\sigma \leftarrow \mathbf{G}^{\text{uf-cma}}_{\mathsf{SS},n}.\textsc{Sign}(i, m)$
Return $\sigma$

Subroutine $\textsc{HSim}(z)$:
If $\text{HT}[z] \neq \bot$ then return $\text{HT}[z]$
$((a_1, a_2), b, c, m) \leftarrow z$ // Parse $z$ as shown where $a_1, a_2, b, c \in \mathbb{G}$ and $m \in \{0,1\}^*$
If ($a_2 = a_1^\beta$ and $c = b^\beta$) then $\text{HT}[z] \leftarrow \mathbf{G}^{\text{uf-cma}}_{\mathsf{SS},n}.\textsc{Hash}(a_1, b, m)$
Else $\text{HT}[z] \leftarrow\$ \mathbb{Z}_p$
Return $\text{HT}[z]$
```

Figure 9: Adversary $A$ for the proof of Theorem 3.2.

Okamoto (Theorem 3.1), the perhaps surprising answer is that security is retained. Again, the assumption made is UF-CMA security of Schnorr. The result is below. We note that, in contrast to Theorem 3.1, here we make crucial use of the ROM, reflected formally in the choices made for the spaces $\mathsf{KW.HS}$ and $\mathsf{SS.HS}$.

As in the Okamoto case, we are able to prove the converse of Theorem 3.2 in Appendix A. (It also relies on the random oracle model.) Thus a reduction to Schnorr is again the best possible result for A-UF-CMA of Katz-Wang.

**Theorem 3.2** *Let $(\mathbb{G}, p, g)$ be a group specification. Let $\mathsf{KW}$ be the associated Katz-Wang signature scheme as above. Let $\mathsf{SS}$ be the Schnorr signature scheme associated to $(\mathbb{G}, p, g)$ and $\mathsf{FUNC}(\mathbb{G} \times \mathbb{G} \times \{0,1\}^*, \mathbb{Z}_p)$. Let $n > 0$ be a number of users. Given an adversary $B$ in game $\mathbf{G}^{\text{a-uf-cma}}_{\mathsf{KW},\mathsf{AGPg}_1^*,n}$ we can construct an adversary $A$ in game $\mathbf{G}^{\text{uf-cma}}_{\mathsf{SS},n}$ such that*

$$\mathbf{Adv}^{\text{a-uf-cma}}_{\mathsf{KW},\mathsf{AGPg}_1^*,n}(B) \leq \mathbf{Adv}^{\text{uf-cma}}_{\mathsf{SS},n}(A) \ . \tag{3}$$

*Adversary $A$ makes at most the same number of signing and hashing queries as $B$, and has running time close to that of $B$.*

The reduction of Eq. (3) is again tight. Nonetheless we note that, in contrast to Theorem 3.1, there will be a tightness loss in reducing A-UF-CMA security of $\mathsf{KW}$, as compared to its UF-CMA security, to algebraic assumptions. Namely, UF-CMA of $\mathsf{KW}$ is shown with a tight reduction to DDH [41]. However, there is no known tight reduction of UF-CMA of Schnorr to DDH. So from Theorem 3.2, the best we can get for A-UF-CMA are looser reductions to DL, MBDL or 2-moment-DL via [1, 15, 50, 53, 54]. Still, Theorem 3.2 is showing the stronger A-UF-CMA security, and thus better-than-advertised security for Katz-Wang in a new setting.

Again, we give in Appendix A a converse to Theorem 3.2, showing that A-UF-CMA of KW implies UF-CMA of Schnorr. This means that A-UF-CMA of KW and UF-CMA of Schnorr are in fact equivalent.

**Proof of Theorem 3.2:** We construct adversary $A$ as shown in Figure 9. In the Schnorr UF-

CMA game, $A$ receives as input verification keys where $\mathbf{X}[i] = g^{\mathbf{s}[i]}$. It picks $\beta$ to compute $h$ and sets up the Katz-Wang verification keys where $\overline{\mathbf{X}}[i] = (\mathbf{X}_g[i], \mathbf{X}_h[i]) = (g^{\mathbf{s}[i]}, h^{\mathbf{s}[i]})$. Now $A$ runs adversary $B$, forwarding signing queries to its own oracle, unchanged. When $B$ makes a query $\textsc{SignSim}(i, m)$ to obtain signature $\sigma = (e, y)$, the $e$ value is not the expected Katz-Wang hash but is instead $\mathbf{G}^{\text{uf-cma}}_{\mathsf{SS},n}.\textsc{Hash}(\mathbf{X}[i], g^r, m)$. The $y$ value is computed identically between Schnorr and Katz-Wang, as $y = (r + e\mathbf{s}[i]) \bmod p$.

However, we ensure that $(e, y)$ is a correct Katz-Wang signature by programming the random oracle. In brief, we enforce that

$$\textsc{HSim}((\mathbf{X}_g[i], \mathbf{X}_h[i]), g^r, h^r, m) = \mathbf{G}^{\text{uf-cma}}_{\mathsf{SS},n}.\textsc{Hash}(\mathbf{X}[i], g^r, m) \qquad (4)$$

when a given $i$, $r$ and $m$ are encountered. Now the Schnorr signature $(e, y)$ is a correct Katz-Wang signature when used with hash function $\textsc{HSim}$, as is $B$'s perspective.

We implement the above condition at lines 8,9 of $\textsc{HSim}$ in Figure 9. Namely, parsing the input $z$ into four group elements and a message on line 8, the check on line 9 ensures that for all $X, r, m$ we have:

$$\textsc{HSim}((X, X^\beta), g^r, g^{\beta r}, m) = \mathbf{G}^{\text{uf-cma}}_{\mathsf{SS},n}.\textsc{Hash}(X, g^r, m) \qquad (5)$$

which in particular implies Eq. (4).

The above ensures that $(m, (e, y))$ is a valid Schnorr signature for $A$ (using $\mathbf{G}^{\text{uf-cma}}_{\mathsf{SS},n}.\textsc{Hash}$) if and only if it is a valid Katz-Wang signature for $B$ (using $\textsc{HSim}$). In particular this means that any forgery output by $B$ may be reused directly by $A$. We finally argue that $\textsc{HSim}$ behaves appropriately like a random oracle to $B$. Note that for any given $a_1, b$ values parsed in $\textsc{HSim}$ there is only one corresponding $a_2$ and only one $c$ that cause line 9 to return. More concisely, the mapping from line-9-passing inputs $((a_1, a_2), b, c, m)$ to $(a_1, b, m)$ is one-to-one, since $\beta$ is fixed and known to $A$. Thus a value $\mathbf{G}^{\text{uf-cma}}_{\mathsf{SS},n}.\textsc{Hash}(a_1, b, m)$ is never returned twice. This, along with lines 7,10 that instantiate the random oracle in all other cases, ensure that all the $\textsc{HSim}$ responses are independent and uniform to $B$. We conclude the proof by noting that $A$ makes the same number of signing queries as $B$, and at most the same number of $\textsc{Hash}$ queries. ▌

## 3.4 Blind, threshold and modern signature schemes

One may justly say that our positive results apply to a few schemes that are not prominent in practice. Nonetheless we posit that resistance to parameter attacks remains a useful security property. GEP signature schemes continue to be designed, including [11, 23, 26, 38]. In one example from 2023 [57], trusted setup is called for to generate the Okamoto parameter $h$ (p. 3 of the full version). The results of this section clarify that this is actually not necessary for Okamoto, which continues to be revisited.

We leave positive results for more recent schemes as an open question. In [23], the authors present three blind signature schemes for a group specification $(\mathbb{G}, p, g)$; all include a random element $W \in \mathbb{G}$ in the public parameters. If one knows $\log_g W$, unforgeability is broken in all three schemes. In [11], when instantiated with the AOMCDH variant, the threshold signature scheme is close to our setting. It uses a group specification $(\mathbb{G}, p, g')$ and random element $g \in \mathbb{G}$. But $g'$ is not used in the scheme, only $g$ is, so while it is syntactically a GEP scheme, it is different than the spirit of our setting. Recent papers also take care to explain parameter generation such as [38, 39]. The goal is to prevent subversion but precomputation attacks continue to apply.

## 4 Encryption under parameter subversion

In this section we define security of public-key encryption against attackers with parameter-related advice. We then consider Cramer-Shoup, in the form where it has parameters and the advice is their discrete logs. We prove that the scheme retains IND-CCA1 security in this setting. With regard to IND-CCA2 in this setting, we have neither a proof nor an attack. We start with the definitions.

### 4.1 Public-key encryption definitions

A PKE scheme $\mathsf{PKE}$ consists of algorithms $\mathsf{PKE.Pg}$ for (honest) parameter generation, $\mathsf{PKE.Kg}$ for key generation, $\mathsf{PKE.Enc}$ for encryption and $\mathsf{PKE.Dec}$ (which is deterministic) for decryption. Public parameters are generated via $\pi \leftarrow\!\!\$\ \mathsf{PKE.Pg}$ and per-user keys via $(ek, dk) \leftarrow\!\!\$\ \mathsf{PKE.Kg}(\pi)$. Encryption takes as input $\pi$ and $ek$ along with a message $m \in \mathsf{PKE.MS}$ to return a ciphertext $C \leftarrow\!\!\$\ \mathsf{PKE.Enc}(\pi, ek, m)$, where $\mathsf{PKE.MS}$ is the message space for $\mathsf{PKE}$. Decryption recovers $m \in \mathsf{PKE.MS} \cup \{\bot\}$ via $m \leftarrow \mathsf{PKE.Dec}(\pi, dk, C)$. Correctness requires that $\mathsf{PKE.Dec}(\pi, dk, \mathsf{PKE.Enc}(\pi, ek, m)) = m$ for all $\pi \in \mathrm{OUT}(\mathsf{PKE.Pg})$, all $(ek, dk) \in \mathrm{OUT}(\mathsf{PKE.Kg}(\pi))$ and all $m \in \mathsf{PKE.MS}$.

We don't, unlike for signatures, include a hash-function space associated to the scheme because our results will not use the ROM. (Cramer-Shoup does use a hash function, but it is only assumed collision-resistant.) Games correspondingly will have no Hash oracle. Games will be parameterized by a number of users $n$ as we continue to be in the multi-user setting.

Usual security notions for public-key encryption are IND-CPA [33], IND-CCA1 [48] and IND-CCA2 [16,27], where public parameters $\pi$ are honestly generated. We use the multi-user setting [14], with a single, global choice of parameters from which users independently generate their own encryption and decryption keys. For all three usual definitions, we give new, advice-based analogues. These are A-IND-CPA, A-IND-CCA1 and A-IND-CCA2.

We give the definitions via the games in Figure 10. The games $\mathbf{G}^{\text{a-ind-cpa}}_{\mathsf{PKE},\mathsf{APg},n}$, $\mathbf{G}^{\text{a-ind-cca1}}_{\mathsf{PKE},\mathsf{APg},n}$ and $\mathbf{G}^{\text{a-ind-cca2}}_{\mathsf{PKE},\mathsf{APg},n}$ are parameterized by an advice-generation algorithm through which the public parameters and advice are generated as $(\pi, ad) \leftarrow\!\!\$\ \mathsf{APg}$. In these advice games, an adversary $A$ is given both $\pi$ and $ad$ by the Init procedure. Otherwise the adversary has access to the same oracles as in the standard security game. In this way, the games capture the adversary's ability to violate security when, in addition to the parameters, it also has advice related to them. As before, in some cases this reflects how parameter generation was done (and possibly subverted), and in other cases models an adversary that computes the advice from the actual parameters (Intermundium-DL).

The games WLOG allow only one challenge Enc query per user. Dec queries are not present in the (A-)IND-CPA games. The (A-)IND-CCA1 games allow any number of Dec queries prior to an Enc query, after which further queries are disallowed for that user. Lastly, in the (A-)IND-CCA2 games, an adversary may make Dec queries at any time, and the only restriction is that Dec not be called on challenge ciphertexts $C_i^*$, once they are defined for users $i$. The different advantages of an adversary $A$ are defined in the natural way, via:

$$\begin{aligned}
\mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{PKE},n}(A) &= 2 \cdot \Pr[\mathbf{G}^{\text{ind-cpa}}_{\mathsf{PKE},n}(A)] - 1 \quad \text{and} \quad \mathbf{Adv}^{\text{a-ind-cpa}}_{\mathsf{PKE},\mathsf{APg},n}(A) = 2 \cdot \Pr[\mathbf{G}^{\text{a-ind-cpa}}_{\mathsf{PKE},\mathsf{APg},n}(A)] - 1\\
\mathbf{Adv}^{\text{ind-cca1}}_{\mathsf{PKE},n}(A) &= 2 \cdot \Pr[\mathbf{G}^{\text{ind-cca1}}_{\mathsf{PKE},n}(A)] - 1 \quad \text{and} \quad \mathbf{Adv}^{\text{a-ind-cca1}}_{\mathsf{PKE},\mathsf{APg},n}(A) = 2 \cdot \Pr[\mathbf{G}^{\text{a-ind-cca1}}_{\mathsf{PKE},\mathsf{APg},n}(A)] - 1\\
\mathbf{Adv}^{\text{ind-cca2}}_{\mathsf{PKE},n}(A) &= 2 \cdot \Pr[\mathbf{G}^{\text{ind-cca2}}_{\mathsf{PKE},n}(A)] - 1 \quad \text{and} \quad \mathbf{Adv}^{\text{a-ind-cca2}}_{\mathsf{PKE},\mathsf{APg},n}(A) = 2 \cdot \Pr[\mathbf{G}^{\text{a-ind-cca2}}_{\mathsf{PKE},\mathsf{APg},n}(A)] - 1 .
\end{aligned}$$

In this section we give results about A-IND-CPA and A-IND-CCA1, and we discuss, but don't give results for, (A-)IND-CCA2.

```
Games $\mathbf{G}^{\text{ind-cpa}}_{\mathsf{PKE},n}$, $\mathbf{G}^{\text{a-ind-cpa}}_{\mathsf{PKE},\mathsf{APg},n}$

INIT:
 1 $d \leftarrow_{\$} \{0,1\}$
 2 $\pi \leftarrow_{\$} \mathsf{PKE.Pg}$ // Game $\mathbf{G}^{\text{ind-cpa}}_{\mathsf{PKE},n}$ only
 3 $(\pi, ad) \leftarrow_{\$} \mathsf{APg}$ // Game $\mathbf{G}^{\text{a-ind-cpa}}_{\mathsf{PKE},\mathsf{APg},n}$ only
 4 For $i = 1 \ldots n$ do $(\mathbf{ek}[i], \mathbf{dk}[i]) \leftarrow_{\$} \mathsf{PKE.Kg}(\pi)$
 5 Return $(\pi, \mathbf{ek})$ // Game $\mathbf{G}^{\text{ind-cpa}}_{\mathsf{PKE},n}$ only
 6 Return $(\pi, \mathbf{ek}, ad)$ // Game $\mathbf{G}^{\text{a-ind-cpa}}_{\mathsf{PKE},\mathsf{APg},n}$ only

ENC$(i, m_0, m_1)$:
 7 If $\mathsf{done}[i]$ then return $\perp$
 8 $C \leftarrow_{\$} \mathsf{PKE.Enc}(\pi, \mathbf{ek}[i], m_d)$
 9 $\mathsf{done}[i] \leftarrow \mathsf{true}$
10 Return $C$

FIN$(d')$:
11 Return $(d' = d)$
```

```
Games $\mathbf{G}^{\text{ind-cca1}}_{\mathsf{PKE},n}$, $\mathbf{G}^{\text{a-ind-cca1}}_{\mathsf{PKE},\mathsf{APg},n}$

INIT:
 1 $d \leftarrow_{\$} \{0,1\}$
 2 $\pi \leftarrow_{\$} \mathsf{PKE.Pg}$ // Game $\mathbf{G}^{\text{ind-cca1}}_{\mathsf{PKE},n}$ only
 3 $(\pi, ad) \leftarrow_{\$} \mathsf{APg}$ // Game $\mathbf{G}^{\text{a-ind-cca1}}_{\mathsf{PKE},\mathsf{APg},n}$ only
 4 For $i = 1 \ldots n$ do $(\mathbf{ek}[i], \mathbf{dk}[i]) \leftarrow_{\$} \mathsf{PKE.Kg}(\pi)$
 5 Return $(\pi, \mathbf{ek})$ // Game $\mathbf{G}^{\text{ind-cca1}}_{\mathsf{PKE},n}$ only
 6 Return $(\pi, \mathbf{ek}, ad)$ // Game $\mathbf{G}^{\text{a-ind-cca1}}_{\mathsf{PKE},\mathsf{APg},n}$ only

ENC$(i, m_0, m_1)$:
 7 If $\mathsf{done}[i]$ then return $\perp$ // One ENC query allowed
 8 $C \leftarrow_{\$} \mathsf{PKE.Enc}(\pi, \mathbf{ek}[i], m_d)$
 9 $\mathsf{done}[i] \leftarrow \mathsf{true}$ // DEC queries now disallowed
10 Return $C$

DEC$(i, C)$:
11 If $\mathsf{done}[i]$ then return $\perp$
12 $m \leftarrow \mathsf{PKE.Dec}(\pi, \mathbf{dk}[i], C)$
13 Return $m$

FIN$(d')$:
14 Return $(d' = d)$
```

```
Games $\mathbf{G}^{\text{ind-cca2}}_{\mathsf{PKE},n}$, $\mathbf{G}^{\text{a-ind-cca2}}_{\mathsf{PKE},\mathsf{APg},n}$

INIT:
 1 $d \leftarrow_{\$} \{0,1\}$
 2 $\pi \leftarrow_{\$} \mathsf{PKE.Pg}$ // Game $\mathbf{G}^{\text{ind-cca2}}_{\mathsf{PKE},n}$ only
 3 $(\pi, ad) \leftarrow_{\$} \mathsf{APg}$ // Game $\mathbf{G}^{\text{a-ind-cca2}}_{\mathsf{PKE},\mathsf{APg},n}$ only
 4 For $i = 1 \ldots n$ do $(\mathbf{ek}[i], \mathbf{dk}[i]) \leftarrow_{\$} \mathsf{PKE.Kg}(\pi)$
 5 Return $(\pi, \mathbf{ek})$ // Game $\mathbf{G}^{\text{ind-cca2}}_{\mathsf{PKE},n}$ only
 6 Return $(\pi, \mathbf{ek}, ad)$ // Game $\mathbf{G}^{\text{a-ind-cca2}}_{\mathsf{PKE},\mathsf{APg},n}$ only

FIN$(d')$:
 7 Return $(d' = d)$

ENC$(i, m_0, m_1)$:
 8 If $\mathsf{done}[i]$ then return $\perp$
 9 $C_i^* \leftarrow_{\$} \mathsf{PKE.Enc}(\pi, \mathbf{ek}[i], m_d)$
10 $\mathsf{done}[i] \leftarrow \mathsf{true}$
11 Return $C_i^*$

DEC$(i, C)$:
12 If $C = C_i^*$ then return $\perp$
13 $m \leftarrow \mathsf{PKE.Dec}(\pi, \mathbf{dk}[i], C)$
14 Return $m$
```

Figure 10: Games defining IND-CPA and A-IND-CPA security (top left), IND-CCA1 and A-IND-CCA1 security (top right) and IND-CCA2 and A-IND-CCA2 security (bottom), all of a public-key encryption scheme PKE. The "A-" games are parameterized by a particular advice-generation algorithm APg. Without loss of generality we restrict attention to adversaries who make one challenge ENC query per user.

## 4.2 Mixed PKE results: Cramer-Shoup

CRAMER-SHOUP ENCRYPTION. Let $(\mathbb{G}, p, g)$ be a group specification. Let $\mathsf{H} : \mathbb{G} \times \mathbb{G} \times \mathbb{G} \to \mathbb{Z}_p$ be a hash function that is assumed collision-resistant. We associate to $(\mathbb{G}, p, g)$ and H the Cramer-Shoup encryption scheme CS whose algorithms are shown in Figure 11. The message space is $\mathsf{CS.MS} = \mathbb{G}$,

```
CS.Pg:
1  $\beta \leftarrow_\$ \mathbb{Z}_p^*$ ; $h \leftarrow g^\beta$
2  Return $h$

CS.Kg($h$):
3  $(x_1, x_2, y_1, y_2, z) \leftarrow_\$ \mathbb{Z}_p^5$
4  $c \leftarrow g^{x_1} h^{x_2}$
5  $d \leftarrow g^{y_1} h^{y_2}$
6  $f \leftarrow g^z$
7  $ek \leftarrow (c, d, f)$
8  $dk \leftarrow (x_1, x_2, y_1, y_2, z)$
9  Return $(ek, dk)$

CS.Enc($h, ek, m$):
10 $k \leftarrow_\$ \mathbb{Z}_p$
11 $(c, d, f) \leftarrow ek$ // Parse $ek$ as shown
12 $u_1 \leftarrow g^k$ ; $u_2 \leftarrow h^k$
13 $e \leftarrow f^k m$
14 $\alpha \leftarrow \mathsf{H}(u_1, u_2, e)$
15 $v \leftarrow c^k d^{k\alpha}$
16 Return $(u_1, u_2, e, v)$

CS.Dec($h, dk, (u_1, u_2, e, v)$):
17 $\alpha \leftarrow \mathsf{H}(u_1, u_2, e)$
18 $(x_1, x_2, y_1, y_2, z) \leftarrow dk$ // Parse $dk$ as shown
19 If $v \neq u_1^{x_1} u_2^{x_2} (u_1^{y_1} u_2^{y_2})^\alpha$ then return $\perp$
20 $m \leftarrow e u_1^{-z}$
21 Return $m$
```

Figure 11: Cramer-Shoup encryption scheme $\mathsf{CS}$ associated to group specification $(\mathbb{G}, p, g)$ and hash function $\mathsf{H}$.

meaning messages are group elements.

This description views the second generator $h$ of the scheme as a public parameter; this has always been a possibility in Cramer-Shoup [25]. In some versions, $h$ is explicitly part of a user's encryption key, and a different $h$ is selected for each user. Scheme $\mathsf{CS1b}$ of [25] is a good example; here $h = g^\beta$ is included in $ek$ while $\beta$ is retained in $dk$ and speeds up decryption. Beyond the Intermundium-DL setting it is thus interesting that knowledge of $\beta$ does not allow one to break security, as we will show.

Delayed-target DDH. We will show A-IND-CCA1 security of $\mathsf{CS}$ under the DT-DDH (delayed-target DDH) assumption. We define it over $n$ users via game $\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}$ on the right side of Figure 12 associated to group specification $(\mathbb{G}, p, g)$. For an adversary $A$, we let $\mathbf{Adv}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A) = 2 \cdot \Pr[\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A)] - 1$ be its advantage. In this game, for each user, an adversary is given a usual DDH challenge $(g^z, T, S)$ and tries to decide whether $S = T^z$ or if $S$ is a random group element (lines 4,5 of Figure 12). However, *prior* to receiving $(T, S)$, the adversary has access to an oracle DH which on input $Y$, returns $Y^z$ (line 8 of the figure). The ordering of queries is enforced by a $\mathsf{done}$ flag; note that without this an adversary could simply call $\mathrm{DH}(T)$ and easily win the game.

In our definitions and in Figure 12 this game is over $n$ users — meaning, specifically, that there is a single challenge bit $d$ and $n$ independent tuples $(\mathbf{Z}[i], \mathbf{T}[i], \mathbf{S}[i])$ — but a standard hybrid argument, provided for completeness in Appendix B, shows that this is implied (non-tightly) by the single-user case.

In referring to this as the delayed-target DDH problem, we use the terminology of Koblitz and Menezes [42]. Delayed-target CDH (under a different name) was first used by Freeman [29, Definition 4.2] to prove security of an identification scheme. Delayed-target DDH (under a different name) was used by Lipmaa [44, Section 4] to prove security of Elgamal. These works all considered the single user $(n = 1)$ case.

We note that if the adversary makes no DH queries, then DT-DDH is equivalent to DDH, over the same number $n$ of users. (The multi-user DDH problem is defined on the left side of Figure 12.)

```
Game $\mathbf{G}^{\mathrm{ddh}}_{\mathbb{G},p,g,n}$

INIT:
  $d \leftarrow\$ \{0,1\}$
  For $i = 1,\ldots,n$ do $\mathbf{z}[i] \leftarrow\$ \mathbb{Z}_p$ ; $\mathbf{Z}[i] \leftarrow g^{\mathbf{z}[i]}$
  For $i = 1,\ldots,n$ do $\mathbf{T}[i] \leftarrow\$ \mathbb{G}$ ; $\mathbf{R}[i] \leftarrow\$ \mathbb{G}$
  If $(d = 1)$ then $\mathbf{S} \leftarrow \mathbf{T}^{\mathbf{z}}$
  If $(d = 0)$ then $\mathbf{S} \leftarrow \mathbf{R}$
  Return $(\mathbf{Z},\mathbf{T},\mathbf{S})$

FIN$(d')$:
  Return $(d' = d)$
```

```
Game $\mathbf{G}^{\mathrm{dt\text{-}ddh}}_{\mathbb{G},p,g,n}$

INIT:
  $d \leftarrow\$ \{0,1\}$
  For $i = 1,\ldots,n$ do $\mathbf{z}[i] \leftarrow\$ \mathbb{Z}_p$ ; $\mathbf{Z}[i] \leftarrow g^{\mathbf{z}[i]}$
  For $i = 1,\ldots,n$ do $\mathbf{T}[i] \leftarrow\$ \mathbb{G}$ ; $\mathbf{R}[i] \leftarrow\$ \mathbb{G}$
  If $(d = 1)$ then $\mathbf{S} \leftarrow \mathbf{T}^{\mathbf{z}}$
  If $(d = 0)$ then $\mathbf{S} \leftarrow \mathbf{R}$
  Return $\mathbf{Z}$

DH$(i, Y)$:
  If done$[i]$ then return $\perp$
  Return $Y^{\mathbf{z}[i]}$

GETTGT$(i)$:
  done$[i] \leftarrow$ true
  Return $(\mathbf{T}[i], \mathbf{S}[i])$

FIN$(d')$:
  Return $(d' = d)$
```

Figure 12: The DDH and DT-DDH (delayed-target decisional Diffie-Hellman) games. The notation $\mathbf{S} \leftarrow \mathbf{T}^{\mathbf{z}}$, over vectors, indicates that if $|\mathbf{T}| = |\mathbf{z}| = n$ then $|\mathbf{S}| = n$ and $\mathbf{S}[i] = \mathbf{T}[i]^{\mathbf{z}[i]}$ for all $1 \le i \le n$.

In particular this means that DT-DDH $\Rightarrow$ DDH, when there are zero DH queries. We will use this in Theorem 4.2 to obtain A-IND-CPA security of Cramer-Shoup under DDH as a corollary of its A-IND-CCA1 security under DT-DDH that we show in Theorem 4.1.

Attacks for delayed-target Diffie-Hellman problems have been described by Joux, Lercier, Naccache and Thomé [37]. (Specifically they attack DT-CDH.) Such attacks take subexponential time for the group of integers modulo a prime, but do not extend to elliptic curve groups.

SECURITY OF CRAMER-SHOUP AGAINST PARAMETER ATTACKS. The Cramer-Shoup PKE scheme CS was shown IND-CCA2-secure under DDH in [25]. Now considering parameter attacks, Theorem 4.1 shows A-IND-CCA1 security. The assumption we make for this is DT-DDH. We leave A-IND-CCA2 security as an interesting open question, to either prove it under a reasonable assumption, or give an attack violating it. In the theorem below, the parameter-generation algorithm for CS is CS.Pg $=$ $\mathsf{GPg}^*_1$, so in considering definitions with advice we will set APg $=$ $\mathsf{AGPg}^*_1$.

Below is the formal statement and proof. In the latter, the adversary strategy is to embed the $Z$ component of the DT-DDH challenge into the $f = g^z$ component of the Cramer-Shoup encryption key. This is different from [25]. The proof refers to the games of Figure 13.

**Theorem 4.1** *Let $(\mathbb{G}, p, g)$ be a group specification and $\mathsf{H} : \mathbb{G} \times \mathbb{G} \times \mathbb{G} \to \mathbb{Z}_p$ a hash function. Let CS be the associated Cramer-Shoup encryption scheme as above. Let $n > 0$ be a number of users. Given an adversary $B$ in game $\mathbf{G}^{\mathrm{a\text{-}ind\text{-}cca1}}_{\mathsf{CS},\mathsf{AGPg}^*_1,n}$ we can construct an adversary $A$ in game $\mathbf{G}^{\mathrm{dt\text{-}ddh}}_{\mathbb{G},p,g,n}$ such that*

$$\mathbf{Adv}^{\mathrm{a\text{-}ind\text{-}cca1}}_{\mathsf{CS},\mathsf{AGPg}^*_1,n}(B) \le 2 \cdot \mathbf{Adv}^{\mathrm{dt\text{-}ddh}}_{\mathbb{G},p,g,n}(A) \ . \qquad (6)$$

*Adversary $A$ makes the same number of* DH *queries as $B$ makes* DEC *queries, at most one* GETTGT *query per user, and has running time close to that of $B$.*

```
Games G_0, G_1

INIT:
  $\delta \leftarrow_\$ \{0,1\}$ ; $\beta \leftarrow_\$ \mathbb{Z}_p^*$ ; $h \leftarrow g^\beta$
  For $i = 1, \ldots, n$ do
      $(x_1, x_2, y_1, y_2) \leftarrow_\$ \mathbb{Z}_p^4$
      $\mathbf{z}[i] \leftarrow_\$ \mathbb{Z}_p$ ; $\mathbf{Z}[i] \leftarrow g^{\mathbf{z}[i]}$
      $c \leftarrow g^{x_1} h^{x_2}$ ; $d \leftarrow g^{y_1} h^{y_2}$
      $\mathbf{ek}[i] \leftarrow (c, d, \mathbf{Z}[i])$ ; $\mathbf{dk}[i] \leftarrow (x_1, x_2, y_1, y_2, \mathbf{z}[i])$
  Return $(h, \mathbf{ek}, \beta)$

ENC$(i, m_0, m_1)$:
  If $\mathsf{done}[i]$ then return $\perp$
  $(c, d, \mathbf{Z}[i]) \leftarrow \mathbf{ek}[i]$
  $t \leftarrow_\$ \mathbb{Z}_p$ ; $u_1 \leftarrow g^t$ ; $u_2 \leftarrow h^t$
  $\mathbf{t}[i] \leftarrow t$ ; $\mathbf{T}[i] \leftarrow u_1$
  $\mathbf{S}[i] \leftarrow \mathbf{Z}[i]^{\mathbf{t}[i]}$  // Game G_0
  $\mathbf{R}[i] \leftarrow_\$ \mathbb{G}$ ; $\mathbf{S}[i] \leftarrow \mathbf{R}[i]$  // Game G_1
  $e \leftarrow \mathbf{S}[i] \cdot m_\delta$
  $\alpha \leftarrow \mathsf{H}(u_1, u_2, e)$
  $v \leftarrow c^t d^{t\alpha}$
  $\mathsf{done}[i] \leftarrow \mathsf{true}$
  Return $(u_1, u_2, e, v)$

DEC$(i, C)$:
  If $\mathsf{done}[i]$ then return $\perp$
  $(x_1, x_2, y_1, y_2, \mathbf{z}[i]) \leftarrow \mathbf{dk}[i]$
  $(u_1, u_2, e, v) \leftarrow C$ ; $\alpha \leftarrow \mathsf{H}(u_1, u_2, e)$
  If $v \neq u_1^{x_1} u_2^{x_2} (u_1^{y_1} u_2^{y_2})^\alpha$ then return $\perp$
  $m \leftarrow e \cdot u_1^{-\mathbf{z}[i]}$
  Return $m$

FIN$(\delta')$:
  Return $(\delta' = \delta)$
```

```
Adversary $A(\mathbf{Z})$:
  $\delta \leftarrow_\$ \{0,1\}$ ; $\beta \leftarrow_\$ \mathbb{Z}_p^*$ ; $h \leftarrow g^\beta$
  For $i = 1, \ldots, n$ do
      $(x_1, x_2, y_1, y_2) \leftarrow_\$ \mathbb{Z}_p^4$
      $c \leftarrow g^{x_1} h^{x_2}$ ; $d \leftarrow g^{y_1} h^{y_2}$
      $\mathbf{ek}[i] \leftarrow (c, d, \mathbf{Z}[i])$ ; $\mathbf{dk}[i] \leftarrow (x_1, x_2, y_1, y_2, \perp)$
  $\delta' \leftarrow B[\text{EncSim}, \text{DecSim}](h, \mathbf{ek}, \beta)$
  If $(\delta' = \delta)$ then return 1
  Else return 0

Subroutine EncSim$(i, m_0, m_1)$:
  If $\mathsf{done}[i]$ then return $\perp$
  $(x_1, x_2, y_1, y_2, \perp) \leftarrow \mathbf{dk}[i]$
  $(\mathbf{T}[i], \mathbf{S}[i]) \leftarrow \text{GetTgt}(i)$
  $u_1 \leftarrow \mathbf{T}[i]$ ; $u_2 \leftarrow \mathbf{T}[i]^\beta$
  $e \leftarrow \mathbf{S}[i] \cdot m_\delta$
  $\alpha \leftarrow \mathsf{H}(u_1, u_2, e)$
  $v \leftarrow u_1^{x_1} u_2^{x_2} (u_1^{y_1} u_2^{y_2})^\alpha$
  $\mathsf{done}[i] \leftarrow \mathsf{true}$
  Return $(u_1, u_2, e, v)$

Subroutine DecSim$(i, (u_1, u_2, e, v))$:
  If $\mathsf{done}[i]$ then return $\perp$
  $(x_1, x_2, y_1, y_2, \perp) \leftarrow \mathbf{dk}[i]$
  $\alpha \leftarrow \mathsf{H}(u_1, u_2, e)$
  If $v \neq u_1^{x_1} u_2^{x_2} (u_1^{y_1} u_2^{y_2})^\alpha$ then return $\perp$
  $w \leftarrow \mathrm{DH}(i, u_1)$
  $m \leftarrow e \cdot w^{-1}$
  Return $m$
```

Figure 13: **Left:** Games for the proof of Theorem 4.1. Lines with comments are only present in the indicated game. **Right:** Adversary $A$ for the proof of Theorem 4.1.

**Proof of Theorem 4.1:** Adversary $A$ is in the right panel of Figure 13. $A$ is in game $\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}$ and receives as input a vector $\mathbf{Z}$ of length $n$. While it will eventually receive the corresponding DT-DDH challenge components $\mathbf{T}, \mathbf{S}$, this occurs in stages, as we will describe. In this proof we refer to the games $G_0, G_1$ in the left panel of Figure 13. Oracles INIT, DEC and FIN are as in game $\mathbf{G}^{\text{a-ind-cca1}}_{\mathsf{CS},\mathsf{AGPg}_1^*,n}$, but $G_0$ and $G_1$ vary in how they respond to ENC queries. We claim that

$$\Pr[\mathbf{G}^{\text{a-ind-cca1}}_{\mathsf{CS},\mathsf{AGPg}_1^*,n}(B)] = \Pr[G_0(B)] \tag{7}$$

$$\Pr[G_0(B)] - \Pr[G_1(B)] \leq \mathbf{Adv}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A) \tag{8}$$

$$\Pr[G_1(B)] = 1/2\ . \tag{9}$$

The above three equations prove Eq. (6), as we can see by rearranging:

$$\begin{aligned}\mathbf{Adv}^{\text{a-ind-cca1}}_{\mathsf{CS},\mathsf{AGPg}_1^*,n}(B) &= 2\cdot\Pr[\mathbf{G}^{\text{a-ind-cca1}}_{\mathsf{CS},\mathsf{AGPg}_1^*,n}(B)]-1\\ &= 2\cdot\Pr[\mathrm{G}_0(B)]-1\\ &\le 2\cdot\Big(\mathbf{Adv}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A)+\Pr[\mathrm{G}_1(B)]\Big)-1\\ &= 2\cdot\Big(\mathbf{Adv}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A)+1/2\Big)-1\\ &= 2\cdot\mathbf{Adv}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A)\ .\end{aligned}$$

We now turn to proving the three numbered equations above.

We start with Eq. (7). This holds because game $\mathrm{G}_0$ is exactly the A-IND-CCA1 game when instantiated with Cramer-Shoup and advice-generation algorithm $\mathsf{AGPg}_1^*$ over $n$ users. There are a few syntactic differences: the challenge bit is called $\delta$, $\mathbf{Z}[i]$ is used in place of $f$ in encryption keys and nonce $t$, rather than $k$, is used during encryption. Nonetheless note that encryption on lines 12,14 computes $e=\mathbf{Z}[i]^{\mathbf{t}[i]}\cdot m_\delta$ which, in this naming, matches Cramer-Shoup computing $e=f^k\cdot m$ in Figure 11.

Next, game $\mathrm{G}_1$ runs line 13 instead of line 12 in Figure 13. That is, $e$ is selected as $e=\mathbf{S}[i]\cdot m_\delta=\mathbf{R}[i]\cdot m_\delta$ for $\mathbf{R}[i]\leftarrow\!\!\$\ \mathbb{G}$. To justify Eq. (8), we first explain adversary $A$ of Figure 13. Initially, $A$ only receives $\mathbf{Z}$ in game $\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}$. At this point, $A$ chooses its own challenge bit $\delta$ and generates all of the parameters and keys for Cramer-Shoup as usual, except, on line 5, for assigning the $\mathbf{Z}[i]$ components of $\mathbf{ek}$ from its input $\mathbf{Z}$, and setting the $\mathbf{z}[i]$ components of $\mathbf{dk}$ to be $\perp$ as they are unknown to $A$. Then $A$ runs $B$, responding to Enc and Dec queries via EncSim and DecSim as shown.

Let us first explain why Dec queries are correctly answered. On input a user $i$ and ciphertext $(u_1,u_2,e,v)$, the checks on lines 20,21 of Figure 13 are the same as in Cramer-Shoup decryption. Now, instead of decrypting $m=e\cdot u_1^{-\mathbf{z}[i]}$ (since $\mathbf{z}[i]$ is unknown), $A$ computes $m=e\cdot\mathrm{DH}(i,u_1)^{-1}$ on lines 22,23 using its DH oracle from game $\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}$. The latter returns precisely $u_1^{\mathbf{z}[i]}$, meaning that $A$ correctly responds to $B$'s decryption queries with $m=e\cdot u_1^{-\mathbf{z}[i]}$.

Next consider the Enc responses, which depend on $A$'s call to GetTgt($i$) on line 11, and in CCA1 games occur after all Dec queries for that user $i$. We claim that if $A$ receives initial input $\mathbf{Z}$ followed by DT-DDH challenge components $(\mathbf{T}[i],\mathbf{S}[i])$ where $\mathbf{S}[i]=\mathbf{T}[i]^{\mathbf{z}[i]}$, then the Enc responses match $\mathrm{G}_0$. Else, if $\mathbf{S}[i]=\mathbf{R}[i]$ for random $\mathbf{R}[i]$, then the Enc responses match $\mathrm{G}_1$. Recall the definition of DT-DDH advantage, such that for challenge bit $d$,

$$\begin{aligned}\mathbf{Adv}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A) &= 2\cdot\Pr[\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A)]-1\\ &= \Pr[\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A)\mid d=1]+\Pr[\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A)\mid d=0]-1\ .\end{aligned}$$

We now claim that

$$\Pr[\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A)\mid d=1]\ge\Pr[\mathrm{G}_0(B)]$$

$$\Pr[\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A)\mid d=0]=1-\Pr[\mathrm{G}_1(B)]$$

and thus that

$$\mathbf{Adv}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A)\ge\Pr[\mathrm{G}_0(B)]-\Pr[\mathrm{G}_1(B)]\ .$$

The above is justified by the fact that if $d=1$ in $A$'s DT-DDH game, then $\mathbf{S}[i]=\mathbf{T}[i]^{\mathbf{z}[i]}$ and $B$

receives encryption responses consistent with $\mathrm{G}_0$. If $B$ correctly guesses $\delta' = \delta$ then $A$ returns 1 (a correct DT-DDH answer) on line 7 of Figure 13. On the other hand, if $d = 0$ in the DT-DDH game, then $\mathbf{S}[i]$ is random and $B$ receives encryption responses consistent with $\mathrm{G}_1$. If $B$ *incorrectly* guesses $\delta'$ then $A$ returns 0 (a correct DT-DDH answer) on line 8. This proves the above and Eq. (8).

Lastly we explain why $\Pr[\mathrm{G}_1(B)] = 1/2$. Here, the game selects $e$ as $\mathbf{R}[i] \cdot m_\delta$ for $\mathbf{R}[i] \leftarrow\!\$\ \mathbb{G}$. Since $\mathbb{G}$ is a group under multiplication (in our notation), $e$ is then a uniformly random element of $\mathbb{G}$. Crucially, $e$ is independent of the challenge bit $\delta$ so adversary $B$'s advantage can only be 1/2. (Note that $e$ is the only encryption component that could depend on $\delta$.)

We conclude by noting that $A$ makes one DH query for each of $B$'s Dec queries and has running time otherwise close to that of $B$. If $B$ makes one Enc query per user then $A$ makes one GetTgt query per user. Moreover, the ordering of the GetTgt queries in DT-DDH is respected as long as the ordering of the Enc queries is respected in A-IND-CCA1. That is, for a particular user $i$, the GetTgt$(i)$ query follows all DH$(i, \cdot)$ queries as long as the Enc$(i, \cdot, \cdot)$ query follows all Dec$(i, \cdot)$ queries. ▌

Rosulek has pointed out that CS remains IND-CPA-secure under DDH to an adversary who knows $\log_g(h)$ [52]. Our definitions allow us to formalize this result; in our language, it says that the scheme is A-IND-CPA-secure under DDH. We note that we can obtain this as a corollary of Theorem 4.1. We state the result below.

**Theorem 4.2** *Let $(\mathbb{G}, p, g)$ be a group specification and $\mathsf{H} : \mathbb{G} \times \mathbb{G} \times \mathbb{G} \to \mathbb{Z}_p$ a hash function. Let* CS *be the associated Cramer-Shoup encryption scheme as above. Let $n > 0$ be a number of users. Given an adversary $B$ in game $\mathbf{G}^{\text{a-ind-cpa}}_{\mathsf{CS},\mathsf{AGPg}^*_1,n}$ we can construct an adversary $A$ in game $\mathbf{G}^{\text{ddh}}_{\mathbb{G},p,g,n}$ such that*

$$\mathbf{Adv}^{\text{a-ind-cpa}}_{\mathsf{CS},\mathsf{AGPg}^*_1,n}(B) \leq 2 \cdot \mathbf{Adv}^{\text{ddh}}_{\mathbb{G},p,g,n}(A)\ . \tag{10}$$

*Adversary $A$ has running time close to that of $B$.*

**Proof of Theorem 4.2:** We can cast $B$ as an adversary for game $\mathbf{G}^{\text{a-ind-cca1}}_{\mathsf{CS},\mathsf{AGPg}^*_1,n}$ that makes no queries to its Dec oracle. Now Theorem 4.1 gives us an adversary in game $\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}$. As per the statement of that theorem, this adversary will make no queries to its DH oracle. This means it can be easily cast as an adversary for game $\mathbf{G}^{\text{ddh}}_{\mathbb{G},p,g,n}$ and the theorem follows. ▌

The two theorems above involve $n$-user (DT-)DDH games. The tight equivalence of multi- and single-user DDH is known [14]. We prove the same (non-tightly) for DT-DDH in Appendix B.

## 5 PAKEs under parameter subversion

A password-authenticated key exchange (PAKE) protocol allows two parties to establish a shared session key using a short secret, the password, which may be drawn from a small set of values. In this section we begin by reviewing PAKE definitions and introduce our security notion for dl-equipped adversaries. In Section 5.2 we give schemes that are totally broken in this setting, and then in contrast, give a proof of security for SPAKE2 in Section 5.3. Regarding practical usage, SPAKE2 has a 2023 RFC [43] and appears in Kerberos [45, 47].

## 5.1 PAKE definitions

We refer to the two parties involved in an execution of the protocol as the client and the server. In the multi-user setting, we denote the set of clients by $\mathcal{C}$ and that of servers by $\mathcal{S}$, which are assumed to be disjoint. Each pair $(\mathsf{C}, \mathsf{S}) \in \mathcal{C} \times \mathcal{S}$ holds a shared password $\mathsf{pw}_{\mathsf{CS}}$. When we refer to a party $\mathsf{P}$, then $\mathsf{P}$ may be either a client or a server. Following [19], each party $\mathsf{P}$ has multiple instances $\tau_{\mathsf{P}}^1, \tau_{\mathsf{P}}^2, \ldots$ and each instance has its own state. We denote the session key space by $\mathcal{K}$. It may depend on global parameters such as the group. We further denote the password space as $\mathcal{PW}$. We continue to let $\pi$ denote public parameters.

A protocol specifies how parties compute the next message to send, as a function of their state and a received message. We let $\mathsf{PAKE.nr}$ be the fixed number of rounds of a protocol $\mathsf{PAKE}$. Such a protocol is often described in a picture such as Figure 14. We only consider 2-round (SPAKE2) or 3-round (KOY) protocols so our definitions are tailored to these cases.

In the below, we recall the game-based security model from Abdalla and Barbosa [2] which builds on the indistinguishability-based security model by Bellare, Pointcheval and Rogaway [17] and its extension to multiple test queries by Abdalla, Fouque and Pointcheval [5]. Recent works on PAKEs often prefer the UC model [3, 6, 13] or to incorporate password-sampling methodologies into game-based definitions [32], but the below is enough to surface distinctions among different schemes' parameter resilience.

For a protocol $\mathsf{PAKE}$ we denote the standard security experiment simply by $\mathbf{G}^{\text{pake}}_{\mathsf{PAKE}}$, and the security experiment for a dl-equipped adversary by $\mathbf{G}^{\text{a-pake}}_{\mathsf{PAKE},\mathsf{APg}}$ for a particular advice-generation algorithm $\mathsf{APg}$. Technically, the property that the following game captures is key-indistinguishability with weak forward secrecy, but we abbreviate this to "pake" in superscripts.

Instance state. The state of an instance $\tau_{\mathsf{P}}^i$ is a tuple $(\mathsf{e}, \mathsf{tr}, \mathsf{sk}, \mathsf{acc})$ where

- $\mathsf{e}$ stores the (secret) ephemeral values chosen by the party in that instance. In our considered schemes, these are elements of $\mathbb{Z}_p$.
- $\mathsf{tr}$ stores the trace (or transcript) of that instance, meaning the client and server name involved in the protocol execution and the messages sent and received by that instance. We assume without loss of generality that the client sends the first message.
- $\mathsf{sk}$ is the accepted session key.
- $\mathsf{acc}$ is a boolean flag that indicates whether the instance has accepted the session key. This can only be set to $\mathsf{acc} = \mathsf{true}$ when the instance has received its last message, as per $\mathsf{PAKE.nr}$.

To access individual components of the state, we write $\tau_{\mathsf{P}}^i.\{\mathsf{e}, \mathsf{tr}, \mathsf{sk}, \mathsf{acc}\}$.

Partnering. Partnering is defined via matching conversations [19]. A client instance $\tau_{\mathsf{C}}^i$ and a server instance $\tau_{\mathsf{S}}^j$ are partnered if and only if

$$\tau_{\mathsf{C}}^i.\mathsf{acc} = \tau_{\mathsf{S}}^j.\mathsf{acc} = \mathsf{true} \quad \text{and} \quad \tau_{\mathsf{C}}^i.\mathsf{tr} = \tau_{\mathsf{S}}^j.\mathsf{tr}\ .$$

Two client instances are never partnered; neither are two server instances. We define a partner predicate $\mathsf{Partner}(\tau_{\mathsf{P}_0}^i, \tau_{\mathsf{P}_1}^j)$ which outputs 1 if the two instances $\tau_{\mathsf{P}_0}^i$ and $\tau_{\mathsf{P}_1}^j$ are partnered and outputs 0 otherwise.

(Standard) security experiment. The security experiment is played between a challenger and an adversary $A$. The challenger draws a random challenge bit $d$ and creates the public parameters $\pi \leftarrow\$ \mathsf{PAKE.Pg}$. Then it outputs the public parameters to $A$. Now $A$ has access to the following oracles:

- $\textsc{Execute}(\mathsf{C}, i, \mathsf{S}, j)$: One complete protocol execution between client instance $\tau_{\mathsf{C}}^{i}$ and server instance $\tau_{\mathsf{S}}^{j}$. This query captures security against passive adversaries, and returns a list of all (public) messages exchanged during the protocol execution.

- $\textsc{SendInit}$, $\textsc{SendResp}$, $\textsc{SendTermInit}$, $\textsc{SendTermResp}$: These four $\textsc{Send}$ oracles model security against active adversaries. Recall that an initiator ($\textsc{Init}$) is a client and a respondent ($\textsc{Resp}$) is a server. Oracle $\textsc{SendInit}(\mathsf{C}, i, \mathsf{S})$ starts a session with client instance $\tau_{\mathsf{C}}^{i}$ and intended partner $\mathsf{S}$ and outputs the first protocol message $m_1$. Oracle $\textsc{SendResp}(\mathsf{S}, i, \mathsf{C}, m_1)$ starts a session with server instance $\tau_{\mathsf{S}}^{i}$, intended partner $\mathsf{C}$ and first protocol message $m_1$, and it outputs the second message $m_2$. Oracle $\textsc{SendTermInit}(\mathsf{C}, i, \mathsf{S}, m_2)$ sends message $m_2$ to instance $\tau_{\mathsf{C}}^{i}$ and outputs $\perp$ (to indicate termination in the case of 2-message protocols) or the third protocol message $m_3$. Similarly $\textsc{SendTermResp}(\mathsf{S}, i, \mathsf{C}, m_3)$, which is only present for 3-message protocols, sends message $m_3$ to $\tau_{\mathsf{S}}^{i}$ and outputs $\perp$.

- $\textsc{Corrupt}(\mathsf{C}, \mathsf{S})$: Outputs the shared password $\mathsf{pw}_{\mathsf{CS}}$ of $\mathsf{C}$ and $\mathsf{S}$.

- $\textsc{Reveal}(\mathsf{P}, i)$: Outputs the session key of instance $\tau_{\mathsf{P}}^{i}$.

- $\textsc{Test}(\mathsf{P}, i)$: A challenge query. Depending on the challenge bit $d$, the experiment outputs either the session key of instance $\tau_{\mathsf{P}}^{i}$ ($d = 0$) or a uniformly random key ($d = 1$). If instance $\tau_{\mathsf{P}}^{i}$ is not fresh (explained below), $\textsc{Test}$ instead returns $\perp$.

- $\textsc{Hash}$: A random oracle, or hash function.

- $\textsc{Fin}(d')$: Returns $\mathsf{true}$ iff $d' = d$.

<u>($\textsc{Advice}$) SECURITY EXPERIMENT.</u> We modify the usual security experiment to generate parameters via $\mathsf{APg}$, which parameterizes the experiment. That is, the challenger draws a random challenge bit $d$ and generates $(\pi, ad) \leftarrow\!\!\$\ \mathsf{APg}$, and outputs both $\pi$ and $ad$ to the adversary $A$. The remaining oracles are as described in the prior paragraph.

<u>FRESHNESS.</u> During the game, we register if a query is allowed in the $\textsc{Test}$ oracle to prevent trivial wins. A query $\textsc{Test}(\mathsf{P}, i)$ is allowed if instance $\tau_{\mathsf{P}}^{i}$ is fresh; in that case $\textsc{Test}$ returns a challenge key, and if not, $\textsc{Test}$ returns $\perp$. In brief, an instance $\tau_{\mathsf{P}}^{i}$ is fresh if it has accepted a session key $\tau_{\mathsf{P}}^{i}.\mathsf{sk}$ which the adversary does not already trivially know. To capture this formally, we define a freshness predicate $\mathsf{Fresh}(\tau_{\mathsf{P}}^{i})$ which takes as input an instance and returns a boolean. (We refer to [2] for the origin of this predicate.) $\mathsf{Fresh}(\tau_{\mathsf{P}}^{i})$ returns $\mathsf{true}$ if and only if

1. $\tau_{\mathsf{P}}^{i}$ accepted, meaning that $\tau_{\mathsf{P}}^{i}.\mathsf{acc} = \mathsf{true}$ and thus that $\tau_{\mathsf{P}}^{i}.\mathsf{sk}$ is defined.

2. There has not been a prior $\textsc{Test}(\mathsf{P}, i)$ nor $\textsc{Reveal}(\mathsf{P}, i)$ query.

3. At least one of the following conditions holds:

   (3.1) $\tau_{\mathsf{P}}^{i}$ was involved in a query to $\textsc{Execute}$, meaning that for some other $\mathsf{P}_1, i_1$ either $\textsc{Execute}(\mathsf{P}, i, \mathsf{P}_1, i_1)$ or $\textsc{Execute}(\mathsf{P}_1, i_1, \mathsf{P}, i)$ returned a non-$\perp$ answer.

   (3.2) There exist at least two partner instances, meaning distinct $i_1, i_2$ and party $\mathsf{P}_1$ such that $\mathsf{Partner}(\tau_{\mathsf{P}}^{i}, \tau_{\mathsf{P}_1}^{i_1}) = \mathsf{true}$ and $\mathsf{Partner}(\tau_{\mathsf{P}}^{i}, \tau_{\mathsf{P}_1}^{i_2}) = \mathsf{true}$.

   (3.3) One fresh partner exists. That is, there is an instance $\tau_{\mathsf{P}_1}^{i_1}$ such that $\mathsf{Partner}(\tau_{\mathsf{P}}^{i}, \tau_{\mathsf{P}_1}^{i_1}) = \mathsf{true}$ and $\mathsf{Fresh}(\tau_{\mathsf{P}_1}^{i_1}) = \mathsf{true}$. Instance $\tau_{\mathsf{P}_1}^{i_1}$ is necessarily fresh due to (3.1) or (3.4).

   (3.4) No partner exists and $\textsc{Corrupt}$ was not queried on the parties $\mathsf{C}, \mathsf{S}$ listed in $\tau_{\mathsf{P}}^{i}.\mathsf{tr}$.

Intuitively, condition (3.1) captures passive eavesdroppers who do not know the ephemeral values of either party. The remaining conditions capture adversaries who make SEND queries, depending on the number of partners of the queried instance. If that number of partners is zero, (3.4) captures an adversary who actively participates on one side but knows neither the password nor the other party's ephemeral values. If there is exactly one partner, (3.3) captures the fact that either both or none of the partnered pair are fresh, as they have the same accepted session key. (One party must be the first to receive a terminal message, and that party is the first to be classified as fresh or not.) Lastly, (3.2) captures adversaries who are able to violate the uniqueness of partner instances, and thus are understood to have broken the PAKE's desired security.

The above definition captures weak forward secrecy since condition (3.1) does not restrict the adversary from calling CORRUPT; such an adversary could learn a password but not ephemeral values. Recall that we denote the (standard) security experiment by $\mathbf{G}^{\mathrm{pake}}_{\mathsf{PAKE}}$ and the security experiment with advice by $\mathbf{G}^{\mathrm{a\text{-}pake}}_{\mathsf{PAKE},\mathsf{APg}}$. For a detailed description of the game in pseudocode, we refer to game $\mathrm{G}_0$ in Appendix C, which is the $\mathbf{G}^{\mathrm{a\text{-}pake}}_{\mathsf{PAKE},\mathsf{APg}}$ game instantiated with protocol $\mathsf{SPAKE2}$. For an adversary $A$ we define the advantages $\mathbf{Adv}^{\mathrm{pake}}_{\mathsf{PAKE}}(A) = 2 \cdot \Pr[\mathbf{G}^{\mathrm{pake}}_{\mathsf{PAKE}}(A)] - 1$ and $\mathbf{Adv}^{\mathrm{a\text{-}pake}}_{\mathsf{PAKE},\mathsf{APg}}(A) = 2 \cdot \Pr[\mathbf{G}^{\mathrm{a\text{-}pake}}_{\mathsf{PAKE},\mathsf{APg}}(A)] - 1$, respectively.

Since passwords are assumed to be chosen from a possibly small set $\mathcal{PW}$, a PAKE is considered secure if the best attack the adversary can do is an online dictionary attack. More concretely, this means that the advantage of the adversary should be negligibly close to $q_{\mathrm{s}}/|\mathcal{PW}|$ when passwords are drawn uniformly and independently from $\mathcal{PW}$, where $q_{\mathrm{s}}$ is the number of SEND queries made by the adversary (to any of the four SEND oracles).

## 5.2 Negative PAKE results: KOY and derivatives

<u>THE KOY PROTOCOL.</u> Katz, Ostrovsky and Yung (KOY) [40] proposed a PAKE protocol in the standard model based on the DDH assumption. It is shown in Figure 14, associated to a group description $(\mathbb{G}, p, g_1)$, hash function $\mathsf{H}$ and signature scheme $\mathsf{Sig}$. The public parameters consist of four randomly chosen group generators $(g_2, h, c, d) \leftarrow_{\$} \mathsf{Gens}(\mathbb{G})^4$. The session key space is the group $\mathbb{G}$.

Follow-up works, most notably the Gennaro-Lindell framework [31], the protocol proposed by Jiang and Gong [36], which simplifies the design of KOY, and its generalization by Groce and Katz [34], use a similar approach and are vulnerable to the same attack as we describe below for KOY.

<u>THE KOY PARAMETER ATTACK.</u> The KOY protocol is vulnerable to parameter attacks; specifically, knowledge of the natural backdoor $\beta = \log_{g_1} h$ allows an attacker to passively learn $g_1^{\mathsf{pw}_{\mathsf{CS}}}$. This enables the attacker to impersonate a client or server without knowing the password $\mathsf{pw}_{\mathsf{CS}}$. We state and justify this in the proposition below, referring to Figure 15. We describe an impersonation of a client, but one could similarly use $g_1^{\mathsf{pw}_{\mathsf{CS}}}$ to impersonate a server. Note, nonetheless, that this does not contradict any security claims of KOY [40], because there it is assumed the adversary does not have any discrete logs of the public parameters.

**Proposition 5.1** *Let $(\mathbb{G}, p, g_1)$ be a group specification. Let $\mathsf{KOY}$ be the associated PAKE protocol as in Figure 14. The adversary $A_{\mathrm{koy}}$ in Figure 15 achieves A-PAKE advantage*

$$\mathbf{Adv}^{\mathrm{a\text{-}pake}}_{\mathsf{KOY},\mathsf{AGPg}_4^*}(A_{\mathrm{koy}}) \geq 1 - 1/p \,. \tag{11}$$

*Adversary $A_{\mathrm{koy}}$ makes one* EXECUTE *and* TEST *query, two* SEND *queries and two* HASH *queries. It takes about the time of a client execution of* $\mathsf{KOY}$.

Public Parameters: $(g_2, h, c, d) \leftarrow\$\ \mathsf{Gens}(\mathbb{G})^4$

| **Client $\mathsf{C}$** | | **Server $\mathsf{S}$** |
|---|---|---|
| Input: $\mathsf{pw}_{\mathsf{CS}} \in \mathbb{Z}_p$ | | Input: $\mathsf{pw}_{\mathsf{CS}} \in \mathbb{Z}_p$ |
| $(vk, sk) \leftarrow\$\ \mathsf{Sig.Kg}$<br>$r_1 \leftarrow\$\ \mathbb{Z}_p$ ; $A \leftarrow g_1^{r_1}$ ; $B \leftarrow g_2^{r_1}$<br>$C \leftarrow h^{r_1} g_1^{\mathsf{pw}_{\mathsf{CS}}}$<br>$\alpha \leftarrow \mathsf{H}(\mathsf{C}, vk, A, B, C)$<br>$D \leftarrow (cd^{\alpha})^{r_1}$ | $\xrightarrow{\mathsf{C}, vk, A, B, C, D}$ | $x_2, y_2, z_2, w_2, r_2 \leftarrow\$\ \mathbb{Z}_p$<br>$\alpha' \leftarrow \mathsf{H}(\mathsf{C}, vk, A, B, C)$<br>$E \leftarrow g_1^{x_2} g_2^{y_2} h^{z_2} (cd^{\alpha'})^{w_2}$<br>$F \leftarrow g_1^{r_2}$ ; $G \leftarrow g_2^{r_2}$ ; $I \leftarrow h^{r_2} g_1^{\mathsf{pw}_{\mathsf{CS}}}$<br>$\gamma \leftarrow \mathsf{H}(\mathsf{S}, E, F, G, I)$ ; $J \leftarrow (cd^{\gamma})^{r_2}$ |
| $x_1, y_1, z_1, w_1 \leftarrow\$\ \mathbb{Z}_p$<br>$\gamma' \leftarrow \mathsf{H}(\mathsf{S}, E, F, G, I)$<br>$K \leftarrow g_1^{x_1} g_2^{y_1} h^{z_1} (cd^{\gamma'})^{w_1}$<br>$\sigma \leftarrow\$\ \mathsf{Sig.Sign}(sk, (\gamma', K))$ | $\xleftarrow{\mathsf{S}, E, F, G, I, J}$ | |
| | $\xrightarrow{K, \sigma}$ | If $\mathsf{Sig.Vfy}(vk, (\gamma, K), \sigma) = 0$<br>  $\mathsf{SK}' \leftarrow\$\ \mathbb{G}$<br>Else:<br>  $C' \leftarrow C / g_1^{\mathsf{pw}_{\mathsf{CS}}}$<br>  $\mathsf{SK}' \leftarrow K^{r_2} A^{x_2} B^{y_2} (C')^{z_2} D^{w_2}$ |
| $I' \leftarrow I / g_1^{\mathsf{pw}_{\mathsf{CS}}}$<br>$\mathsf{SK} \leftarrow E^{r_1} F^{x_1} G^{y_1} (I')^{z_1} J^{w_1}$ | | |
| Output: $\mathsf{SK}$ | | Output: $\mathsf{SK}'$ |

Figure 14: The $\mathsf{KOY}$ protocol associated to group specification $(\mathbb{G}, p, g_1)$, hash function $\mathsf{H}$ and signature scheme $\mathsf{Sig}$.

```
Adversary $A_{\mathrm{koy}}(\pi, ad)$:
1  $(g_2, h, c, d) \leftarrow \pi$ ; $(\beta_1, \beta_2, \beta_3, \beta_4) \leftarrow ad$
2  $\beta \leftarrow \beta_2$ // In fact, we only need one dl
3  Pick any client $\mathsf{C}$ and server $\mathsf{S}$ ; $(vk', sk') \leftarrow\$ \mathsf{Sig.Kg}$
4  $\mathtt{tr} \leftarrow \textsc{Execute}(\mathsf{C}, 0, \mathsf{S}, 0)$
5  $(vk, A, B, C, D, E, F, G, I, J, K, \sigma) \leftarrow \mathtt{tr}$
6  $P \leftarrow C \cdot A^{-\beta}$ // $P = g_1^{\mathsf{pw}_{\mathsf{CS}}}$
7  Compute message $m_1 \leftarrow (\mathsf{C}, vk', A', B', C', D')$ as usual,
       except with $C' \leftarrow h^{r_1} \cdot P$, and ensuring that $A' \neq A$.
8  $(\mathsf{S}, E', F', G', I', J') \leftarrow \textsc{SendResp}(\mathsf{S}, 1, \mathsf{C}, m_1)$
9  Compute message $m_3 \leftarrow (K', \sigma')$ as usual
10 $\textsc{SendTermResp}(\mathsf{S}, 1, \mathsf{C}, m_3)$
11 $I'' \leftarrow I'/P$ ; $\mathsf{SK}^* \leftarrow (E')^{r_1} (F')^{x_1} (G')^{y_1} (I'')^{z_1} (J')^{w_1}$
12 $\mathsf{SK} \leftarrow \textsc{Test}(\mathsf{S}, 1)$
13 If $(\mathsf{SK} = \mathsf{SK}^*)$ then return 0 else return 1
```

Figure 15: Adversary $A_{\mathrm{koy}}$ which breaks A-PAKE security of the $\mathsf{KOY}$ protocol. When a step occurs "as usual" it follows the code in Figure 14.

**Proof of Proposition 5.1:** In the $\mathbf{G}^{\text{a-pake}}_{\mathsf{KOY},\mathsf{AGPg}_4^*}$ security experiment, adversary $A_{\mathrm{koy}}$ executes the steps shown in Figure 15. On line 1 it receives public parameters $(g_2, h, c, d)$ and additionally the advice $(\beta_1, \beta_2, \beta_3, \beta_4)$. The attack only requires one discrete log; we use $\beta = \beta_2$. On lines 3-6 the adversary passively eavesdrops on an instance 0 execution. This is enough to learn $P = C \cdot A^{-\beta} = g_1^{\mathsf{pw}_{\mathsf{CS}}}$, which is the critical step of the attack. After line 6, $A_{\text{koy}}$ impersonates the client to server instance 1 and computes the correct session key $\mathsf{SK}^*$ on line 11. On lines 12,13 it compares $\mathsf{SK}^*$ to a challenge key $\mathsf{SK}$, to determine the game's challenge bit.

Adversary $A_{\text{koy}}$ succeeds with probability $1 - 1/p$, where $p$ is the order of $\mathbb{G}$. The latter term is because line 13 passes with probability $1/p$ on a random key. Otherwise, $A_{\text{koy}}$ learns the correct session key $\mathsf{SK}^*$ and can thus determine the challenge bit. This does require that $\textsc{Test}(\mathsf{S}, 1)$ on line 12 is run on a fresh instance, meaning that all three freshness conditions hold. (1) is true because instance $\tau_{\mathsf{S}}^1$ would have accepted after receiving both correct protocol messages $m_1, m_3$. (2) is true because $\tau_{\mathsf{S}}^1$ had not previously been queried to $\textsc{Test}$ or $\textsc{Reveal}$. (3.4) holds because no partner for $\tau_{\mathsf{S}}^1$ exists and $\textsc{Corrupt}$ was not queried; ensuring $A' \neq A$ on line 7 guarantees that $\tau_{\mathsf{C}}^0$ is not partnered to $\tau_{\mathsf{S}}^1$. ∎

While we are not sure if such an attack on KOY has already been pointed out, we provide it to contrast the following result, which seems to be a unique property to SPAKE2 among PAKE protocols that use group-element parameters.

### 5.3 Mixed PAKE results: SPAKE2 and derivatives

In contrast to KOY, we next show that SPAKE2 retains some security against parameter attacks. On the negative side, an offline dictionary attack applies, so the usual PAKE guarantee is not achieved. Nonetheless, we prove that this is the best possible attack, concluding that SPAKE2 retains security if passwords are high-entropy.

THE SPAKE2 PROTOCOL. We recall the protocol in Figure 16. It was originally proposed by Abdalla and Pointcheval [7] and different variants have been considered in the literature, such as [51]. It is also published as a draft standard [43]. In addition to the group specification $(\mathbb{G}, p, g)$, the protocol uses two public group elements $M$, $N$ and a hash function $\mathsf{H} : \{0,1\}^* \to \mathcal{K}$ which maps to the key space $\mathcal{K}$. We consider passwords to be elements in $\mathbb{Z}_p$; in practice, these may be obtained by hashing the actual password. We set the hash space to be $\mathsf{SPAKE2.HS} = \mathsf{FUNC}(\{0,1\}^*, \mathcal{K})$, indicating that our positive result (Theorem 5.3) will rely on the ROM.

OFFLINE DICTIONARY ATTACK. We describe the attack in Figure 17, letting $M = g^m$ and $N = g^n$ so the natural advice is $(m, n)$. Adversary $A_{\text{dict}}^q$ is in game $\mathbf{G}_{\mathsf{SPAKE2},\mathsf{AGPg}_2}^{\text{a-pake}}$. We claim that if $A_{\text{dict}}^q$ tests the correct password during its dictionary attack (over $q$ password guesses), then it will be able to compute the correct session key and win the A-PAKE game. We explain this in detail in the following proposition.

**Proposition 5.2** *Let $(\mathbb{G}, p, g)$ be a group specification. Let* SPAKE2 *be the associated PAKE protocol as in Figure 16 with password space $\mathcal{PW} \subseteq \mathbb{Z}_p$ and key space $\mathcal{K}$. The adversary $A_{\text{dict}}^q$ in Figure 17 achieves A-PAKE advantage*

$$\mathbf{Adv}_{\mathsf{SPAKE2},\mathsf{AGPg}_2}^{\text{a-pake}}(A_{\text{dict}}^q) \geq q\left(\frac{1}{|\mathcal{PW}|} - \frac{1}{|\mathcal{K}|}\right) , \tag{12}$$

*assuming passwords are chosen uniformly from $\mathcal{PW}$. Adversary $A_{\text{dict}}^q$ makes $q$ queries to* $\textsc{Hash}$, *one* $\textsc{Send}$ *and one* $\textsc{Test}$ *query. It takes about the time of $q$ client executions of* SPAKE2.

The above advantage equation depends on password entropy. If $\mathcal{PW} = \mathbb{Z}_p$ the bound becomes $q(1/p - 1/|\mathcal{K}|)$. However if passwords are human-chosen and $|\mathcal{PW}| \ll |\mathcal{K}|$, the advantage could become greater. Note also that this is an offline attack, as $A_{\text{dict}}^q$ makes $q$ $\textsc{Hash}$ queries but only a single initial $\textsc{Send}$ query and final $\textsc{Test}$ query. We now proceed to the proof.

Public Parameters: $(M, N) \leftarrow_{\$} \mathbb{G}^2$

| **Client $\mathsf{C}$** | | **Server $\mathsf{S}$** |
|---|---|---|
| Input: $\mathsf{pw}_{\mathsf{CS}} \in \mathbb{Z}_p$ | | Input: $\mathsf{pw}_{\mathsf{CS}} \in \mathbb{Z}_p$ |
| $x \leftarrow_{\$} \mathbb{Z}_p$ | | $y \leftarrow_{\$} \mathbb{Z}_p$ |
| $X^* \leftarrow g^x \cdot M^{\mathsf{pw}_{\mathsf{CS}}}$ | $\xrightarrow{\mathsf{C}, X^*}$ | $Y^* \leftarrow g^y \cdot N^{\mathsf{pw}_{\mathsf{CS}}}$ |
| | $\xleftarrow{\mathsf{S}, Y^*}$ | |
| $Y \leftarrow Y^*/N^{\mathsf{pw}_{\mathsf{CS}}}$ ; $Z \leftarrow Y^x$ | | $X \leftarrow X^*/M^{\mathsf{pw}_{\mathsf{CS}}}$ ; $Z' \leftarrow X^y$ |
| $\mathsf{SK} \leftarrow \mathsf{H}(\mathsf{C}, \mathsf{S}, \mathsf{pw}_{\mathsf{CS}}, X^*, Y^*, Z)$ | | $\mathsf{SK}' \leftarrow \mathsf{H}(\mathsf{C}, \mathsf{S}, \mathsf{pw}_{\mathsf{CS}}, X^*, Y^*, Z')$ |
| Output: $\mathsf{SK}$ | | Output: $\mathsf{SK}'$ |

Figure 16: The $\mathsf{SPAKE2}$ protocol associated to group specification $(\mathbb{G}, p, g)$, with hash function $\mathsf{H} \in \mathsf{SPAKE2.HS}$.

**Proof of Proposition 5.2:** Adversary $A^q_{\text{dict}}$ is in Figure 17. It is initialized with public parameters $(M, N)$ and advice $(m, n)$ where $g^m = M$ and $g^n = N$. It is acting as a client, sending the first protocol message $m_1$ on line 5 and there receiving the second protocol message $m_2 = (\mathsf{S}, Y^*)$. On line 6 it requests a challenge key $\mathsf{SK}$, where instance $\tau^0_{\mathsf{S}}$ is fresh due to condition (3.4), and on line 7 begins its dictionary attack. We claim that if the adversary tests the correct password during this loop, that it will compute $Z^*$ on line 8 which matches $Z'$ computed by the server. From there it can compute the correct session key and win the game.

Let us explain in more detail. Recall that the server in SPAKE2 computes $Z' = X^y = (X^*/M^{\mathsf{pw}_{\mathsf{CS}}})^y$ and sends $Y^* = g^{\bar{y}} = g^{y+n\cdot\mathsf{pw}_{\mathsf{CS}}}$. On the correct password $\mathsf{pw}_{\mathsf{CS}}$ this means:

$$Z' = \left(\frac{g^{\bar{x}}}{g^{m\cdot\mathsf{pw}_{\mathsf{CS}}}}\right)^y = g^{(\bar{x}-m\cdot\mathsf{pw}_{\mathsf{CS}})y} = g^{(\bar{x}-m\cdot\mathsf{pw}_{\mathsf{CS}})(\bar{y}-n\cdot\mathsf{pw}_{\mathsf{CS}})}$$
$$= g^{\bar{y}(\bar{x}-m\cdot\mathsf{pw}_{\mathsf{CS}})} \cdot g^{-n\cdot\mathsf{pw}_{\mathsf{CS}}(\bar{x}-m\cdot\mathsf{pw}_{\mathsf{CS}})} = (Y^*)^{\bar{x}-m\cdot\mathsf{pw}_{\mathsf{CS}}} \cdot g^{n\cdot m\cdot\mathsf{pw}^2_{\mathsf{CS}}-\bar{x}\cdot n\cdot\mathsf{pw}_{\mathsf{CS}}}\ .$$

The last expression is what $A^q_{\text{dict}}$ computes on line 8, and thus if $\mathsf{pw}_{\mathsf{CS}}$ is encountered during the iteration, the true $Z^* = Z'$ will be computed. The remaining values $(\mathsf{C}, \mathsf{S}, X^*, Y^*)$ are public so the true $\mathsf{SK}^*$ is computed on line 9.

Suppose that the challenge $\mathsf{SK}$ is the true key, meaning the challenge bit is $d = 0$. Now $A^q_{\text{dict}}$ will output 0 as long as it encounters the correct password (at least). This probability is $q/|\mathcal{PW}|$ for a uniformly random password. If the challenge bit is $d = 1$, $A^q_{\text{dict}}$ will incorrectly return 0 if the random $\mathsf{SK}$ happens to match any of the values $\mathsf{SK}^*$ computed on line 9. Since $\mathsf{SK}$ is uniformly random in $\mathcal{K}$ this probability is at most $q/|\mathcal{K}|$. Combining the two cases yields Eq. (12). ▌

The above shows that offline dictionary attacks can be performed by an attacker with $m, n$. But the above attack is not necessarily efficient, as the attack on KOY was. In particular, if passwords are high-entropy or uniformly random elements of $\mathbb{Z}_p$, the above is infeasibly expensive. Can we prove that SPAKE2 retains some security in this case? In other words, is $A^q_{\text{dict}}$ the best possible attack?

PARTIAL SECURITY OF SPAKE2 AGAINST PARAMETER ATTACKS. The above allows an attacker to brute-force the value $Z'$ by iterating through the password space. Since in the protocol $Z'$ is used as an input to a hash function (modeled as a random oracle), we can "measure" the complexity

```
Adversary $A^q_{\text{dict}}(\pi, ad)$:
$(M, N) \leftarrow \pi$ ; $(m, n) \leftarrow ad$
Pick any client $\mathsf{C}$ and server $\mathsf{S}$
$\bar{x} \leftarrow_{\$} \mathbb{Z}_p$ ; $X^* \leftarrow g^{\bar{x}}$
$m_1 \leftarrow (\mathsf{C}, X^*)$
$(\mathsf{S}, Y^*) \leftarrow \textsc{SendResp}(\mathsf{S}, 0, \mathsf{C}, m_1)$
$\mathsf{SK} \leftarrow \textsc{Test}(\mathsf{S}, 0)$
For $q$ candidate passwords $\mathsf{pw} \in \mathbb{Z}_p$ do:
    $Z^* \leftarrow (Y^*)^{\bar{x} - m \cdot \mathsf{pw}} \cdot g^{n \cdot m \cdot \mathsf{pw}^2 - \bar{x} \cdot n \cdot \mathsf{pw}}$
    $\mathsf{SK}^* \leftarrow \textsc{Hash}(\mathsf{C}, \mathsf{S}, \mathsf{pw}, X^*, Y^*, Z^*)$
    If $(\mathsf{SK}^* = \mathsf{SK})$ then return 0
Return 1
```

Figure 17: Adversary $A^q_{\text{dict}}$ which performs an offline dictionary attack (of $q$ password attempts) on the A-PAKE security of $\mathsf{SPAKE2}$.

```
Game $\mathbf{G}^{\text{scdh}}_{\mathbb{G},p,g,n}$

Init:
For $i = 1, \ldots, n$ do $x_i \leftarrow_{\$} \mathbb{Z}_p$ ; $X_i \leftarrow g^{x_i}$
For $i = 1, \ldots, n$ do $Y_i \leftarrow_{\$} \mathbb{G}$
Return $(X_1, \ldots, X_n, Y_1, \ldots, Y_n)$

DDH$(i, Y', Z)$:
If $Z = (Y')^{x_i}$ then return 1
Else return 0

Fin$(i, Z')$:
If $Z' = (Y_i)^{x_i}$ then return 1
Else return 0
```

Figure 18: The $n$-user strong CDH problem (SCDH) for group specification $(\mathbb{G}, p, g)$.

of the attack. Put differently, we can give a concrete security bound in the game-based model, as captured in the following theorem. It involves the strong CDH (SCDH) assumption [4] which we give over $n$ users in Figure 18. The term in the theorem below involves the $n$-user strong CDH problem with $n = q_{\text{e}}$, but this is tightly equivalent to the single-user version. We denote the former game by $\mathbf{G}^{\text{scdh}}_{\mathbb{G},p,g,n}$. An adversary $A$ has advantage $\mathbf{Adv}^{\text{scdh}}_{\mathbb{G},p,g,n}(A) = \Pr[\mathbf{G}^{\text{scdh}}_{\mathbb{G},p,g,n}(A)]$. We now proceed to the theorem statement.

**Theorem 5.3** *Let $(\mathbb{G}, p, g)$ be a group specification. Let $\mathsf{SPAKE2}$ be the associated PAKE protocol as in Figure 16 with password space $\mathcal{PW} \subseteq \mathbb{Z}_p$. Given an adversary $A$ in game $\mathbf{G}^{\text{a-pake}}_{\mathsf{SPAKE2},\mathsf{AGPg}_2}$ that makes at most $q_{\text{h}}$ Hash queries, $q_{\text{e}}$ Execute queries and $q_{\text{s}}$ Send queries, we can construct an adversary $B$ in game $\mathbf{G}^{\text{scdh}}_{\mathbb{G},g,p,q_{\text{e}}}$ such that*

$$\mathbf{Adv}^{\text{a-pake}}_{\mathsf{SPAKE2},\mathsf{AGPg}_2}(A) \leq \frac{2q_{\text{h}}}{|\mathcal{PW}|} + 2 \cdot \mathbf{Adv}^{\text{scdh}}_{\mathbb{G},g,p,q_{\text{e}}}(B) + \frac{2(q_{\text{s}} + q_{\text{e}})^2}{p} , \tag{13}$$

*assuming passwords are chosen uniformly from $\mathcal{PW}$. Adversary $B$ makes at most $q_{\text{h}}$ queries to its*

*DDH oracle, and has running time close to that of $A$.*

Note that the offline dictionary attack is captured by the first term in the bound. Hence, the above does not match the desired PAKE bound of $q_s/|\mathcal{PW}|$ that captures online attacks. However, if passwords are chosen from a high-entropy distribution, this term becomes negligible. This is the "partial security" that we claim. The proof of Theorem 5.3 is in Appendix C.

Per-user parameters. We conclude our discussion of SPAKE2 by noting the RFC [43] mentions the possibility of using per-user $M$ and $N$ to avoid concerns that an attacker could break authentication by learning only $m, n$. (Secret $m, n$ is described as needed for the security proof, but an attack is not described.) Our analysis shows that while SPAKE2 is indeed subject to offline dictionary attacks, the situation is not as bad as for the KOY family of protocols. Deriving $M$ and $N$ via hashing the client and server identity is a solution that can mitigate precomputation attacks from a theoretical standpoint. However, it may not be easily implemented in practice when client or server identities are not available or need to be exchanged in advance, therefore introducing additional rounds.

## Acknowledgments

We thank the anonymous reviewers for their feedback and suggestions.

## References

[1] M. Abdalla, J. H. An, M. Bellare, and C. Namprempre. From identification to signatures via the Fiat-Shamir transform: Minimizing assumptions for security and forward-security. In L. R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 418–433. Springer, Berlin, Heidelberg, Apr. / May 2002. (Cited on 4, 11, 15.)

[2] M. Abdalla and M. Barbosa. Perfect forward security of SPAKE2. Cryptology ePrint Archive, Report 2019/1194, 2019. (Cited on 24, 25, 41, 42.)

[3] M. Abdalla, M. Barbosa, T. Bradley, S. Jarecki, J. Katz, and J. Xu. Universally composable relaxed password authenticated key exchange. In D. Micciancio and T. Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 278–307. Springer, Cham, Aug. 2020. (Cited on 24.)

[4] M. Abdalla, M. Bellare, and P. Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In D. Naccache, editor, *CT-RSA 2001*, volume 2020 of *LNCS*, pages 143–158. Springer, Berlin, Heidelberg, Apr. 2001. (Cited on 5, 30.)

[5] M. Abdalla, P.-A. Fouque, and D. Pointcheval. Password-based authenticated key exchange in the three-party setting. In S. Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 65–84. Springer, Berlin, Heidelberg, Jan. 2005. (Cited on 24.)

[6] M. Abdalla, B. Haase, and J. Hesse. Security analysis of CPace. In M. Tibouchi and H. Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 711–741. Springer, Cham, Dec. 2021. (Cited on 24.)

[7] M. Abdalla and D. Pointcheval. Simple password-based encrypted key exchange protocols. In A. Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 191–208. Springer, Berlin, Heidelberg, Feb. 2005. (Cited on 2, 4, 28.)

[8] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, and P. Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In I. Ray, N. Li, and C. Kruegel, editors, *ACM CCS 2015*, pages 5–17. ACM Press, Oct. 2015. (Cited on 6.)

[9] B. Auerbach, M. Bellare, and E. Kiltz. Public-key encryption resistant to parameter subversion and its realization from efficiently-embeddable groups. In M. Abdalla and R. Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 348–377. Springer, Cham, Mar. 2018. (Cited on 5.)

[10] B. Auerbach, F. Giacon, and E. Kiltz. Everybody's a target: Scalability in public-key encryption. In A. Canteaut and Y. Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 475–506. Springer, Cham, May 2020. (Cited on 5.)

[11] R. Bacho, J. Loss, S. Tessaro, B. Wagner, and C. Zhu. Twinkle: Threshold signatures from DDH with full adaptive security. In M. Joye and G. Leander, editors, *EUROCRYPT 2024, Part I*, volume 14651 of *LNCS*, pages 429–459. Springer, Cham, May 2024. (Cited on 16.)

[12] M. Backendal, M. Bellare, J. Sorrell, and J. Sun. The Fiat-Shamir zoo: Relating the security of different signature variants. In N. Gruschka, editor, *NordSec 2018*, pages 154–170, Cham, 2018. Springer International Publishing. (Cited on 11.)

[13] M. Barbosa, K. Gellert, J. Hesse, and S. Jarecki. Bare PAKE: Universally composable key exchange from just passwords. In L. Reyzin and D. Stebila, editors, *CRYPTO 2024, Part II*, volume 14921 of *LNCS*, pages 183–217. Springer, Cham, Aug. 2024. (Cited on 24.)

[14] M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In B. Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 259–274. Springer, Berlin, Heidelberg, May 2000. (Cited on 10, 17, 23, 38.)

[15] M. Bellare and W. Dai. The multi-base discrete logarithm problem: Tight reductions and non-rewinding proofs for Schnorr identification and signatures. In K. Bhargavan, E. Oswald, and M. Prabhakaran, editors, *INDOCRYPT 2020*, volume 12578 of *LNCS*, pages 529–552. Springer, Cham, Dec. 2020. (Cited on 4, 11, 12, 15.)

[16] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In H. Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 26–45. Springer, Berlin, Heidelberg, Aug. 1998. (Cited on 17.)

[17] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In B. Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, Berlin, Heidelberg, May 2000. (Cited on 24.)

[18] M. Bellare, T. Ristenpart, and S. Tessaro. Multi-instance security and its application to password-based cryptography. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 312–329. Springer, Berlin, Heidelberg, Aug. 2012. (Cited on 5.)

[19] M. Bellare and P. Rogaway. Entity authentication and key distribution. In D. R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249. Springer, Berlin, Heidelberg, Aug. 1994. (Cited on 24.)

[20] M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Berlin, Heidelberg, May / June 2006. (Cited on 7.)

[21] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang. High-speed high-security signatures. In B. Preneel and T. Takagi, editors, *CHES 2011*, volume 6917 of *LNCS*, pages 124–142. Springer, Berlin, Heidelberg, Sept. / Oct. 2011. (Cited on 11.)

[22] D. J. Bernstein, T. Lange, and R. Niederhagen. Dual EC: A standardized back door. In P. Ryan, D. Naccache, and J.-J. Quisquater, editors, *The New Codebreakers*. Springer, Heidelberg, 2016. (Cited on 2, 6.)

[23] R. Chairattana-Apirom, S. Tessaro, and C. Zhu. Pairing-free blind signatures from CDH assumptions. In L. Reyzin and D. Stebila, editors, *CRYPTO 2024, Part I*, volume 14920 of *LNCS*, pages 174–209. Springer, Cham, Aug. 2024. (Cited on 16.)

[24] A. Commeine and I. Semaev. An algorithm to solve the discrete logarithm problem with the number field sieve. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 174–190. Springer, Berlin, Heidelberg, Apr. 2006. (Cited on 6.)

[25] R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003. (Cited on 2, 4, 5, 19, 20.)

[26] S. Das and L. Ren. Adaptively secure BLS threshold signatures from DDH and co-CDH. In L. Reyzin and D. Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 251–284. Springer, Cham, Aug. 2024. (Cited on 16.)

[27] D. Dolev, C. Dwork, and M. Naor. Nonmalleable cryptography. *SIAM Journal on Computing*, 30(2):391–437, 2000. (Cited on 17.)

[28] E. Eaton and D. Stebila. The "quantum annoying" property of password-authenticated key exchange protocols. In J. H. Cheon and J.-P. Tillich, editors, *Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021*, pages 154–173. Springer, Cham, 2021. (Cited on 5.)

[29] D. Freeman. Pairing-based identification schemes. Cryptology ePrint Archive, Report 2005/336, 2005. (Cited on 5, 19.)

[30] G. Fuchsbauer and M. Wolf. Concurrently secure blind schnorr signatures. In M. Joye and G. Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 124–160. Springer, Cham, May 2024. (Cited on 11.)

[31] R. Gennaro and Y. Lindell. A framework for password-based authenticated key exchange. In E. Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 524–543. Springer, Berlin, Heidelberg, May 2003. (Cited on 4, 26.)

[32] K. Gjøsteen. Password-authenticated key exchange and applications. Cryptology ePrint Archive, Report 2024/1057, 2024. (Cited on 24.)

[33] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984. (Cited on 17.)

[34] A. Groce and J. Katz. A new framework for efficient password-based authenticated key exchange. In E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, editors, *ACM CCS 2010*, pages 516–525. ACM Press, Oct. 2010. (Cited on 4, 26.)

[35] R. Impagliazzo. A personal view of average-case complexity. In *Proceedings of the Tenth Annual Structure in Complexity Theory Conference, Minneapolis, Minnesota, USA, June 19-22, 1995*, pages 134–147. IEEE Computer Society, 1995. (Cited on 3.)

[36] S. Jiang and G. Gong. Password based key exchange with mutual authentication. In H. Handschuh and A. Hasan, editors, *SAC 2004*, volume 3357 of *LNCS*, pages 267–279. Springer, Berlin, Heidelberg, Aug. 2004. (Cited on 4, 26.)

[37] A. Joux, R. Lercier, D. Naccache, and E. Thomé. Oracle-assisted static Diffie-Hellman is easier than discrete logarithms. In M. G. Parker, editor, *12th IMA International Conference on Cryptography and Coding*, volume 5921 of *LNCS*, pages 351–367. Springer, Berlin, Heidelberg, Dec. 2009. (Cited on 20.)

[38] J. Kastner, K. Nguyen, and M. Reichle. Pairing-free blind signatures from standard assumptions in the ROM. In L. Reyzin and D. Stebila, editors, *CRYPTO 2024, Part I*, volume 14920 of *LNCS*, pages 210–245. Springer, Cham, Aug. 2024. (Cited on 16.)

[39] S. Katsumata, M. Reichle, and Y. Sakai. Practical round-optimal blind signatures in the ROM from standard assumptions. In J. Guo and R. Steinfeld, editors, *ASIACRYPT 2023, Part II*, volume 14439 of *LNCS*, pages 383–417. Springer, Singapore, Dec. 2023. (Cited on 16.)

[40] J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. In B. Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 475–494. Springer, Berlin, Heidelberg, May 2001. (Cited on 3, 4, 26.)

[41] J. Katz and N. Wang. Efficiency improvements for signature schemes with tight security reductions. In S. Jajodia, V. Atluri, and T. Jaeger, editors, *ACM CCS 2003*, pages 155–164. ACM Press, Oct. 2003. (Cited on 2, 4, 5, 14, 15.)

[42] N. Koblitz and A. Menezes. Another look at non-standard discrete log and Diffie-Hellman problems. *Journal of Mathematical Cryptology*, 2(4):311–326, 2008. (Cited on 5, 19.)

[43] W. Ladd. RFC 9382: SPAKE2, a password-authenticated key exchange. https://datatracker.ietf.org/doc/rfc9382/, Sept. 2023. (Cited on 2, 4, 6, 23, 28, 31.)

[44] H. Lipmaa. On the CCA1-security of Elgamal and Damgård's Elgamal. In X. Lai, M. Yung, and D. Lin, editors, *Information Security and Cryptology*, pages 18–35, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. (Cited on 5, 19.)

[45] N. McCallum, S. Sorce, R. Harwood, and G. Hudson. RFC 9588: Kerberos simple password-authenticated key exchange (SPAKE) pre-authentication. https://datatracker.ietf.org/doc/rfc9588/, Aug. 2024. (Cited on 23.)

[46] A. Menezes and N. P. Smart. Security of signature schemes in a multi-user setting. *DCC*, 33(3):261–274, 2004. (Cited on 10.)

[47] MIT Kerberos documentation: SPAKE preauthentication. https://web.mit.edu/kerberos/krb5-1.21/doc/admin/spake.html. (Cited on 23.)

[48] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd ACM STOC*, pages 427–437. ACM Press, May 1990. (Cited on 17.)

[49] T. Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In E. F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 31–53. Springer, Berlin, Heidelberg, Aug. 1993. (Cited on 2, 3, 4, 11.)

[50] D. Pointcheval and J. Stern. Security proofs for signature schemes. In U. M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 387–398. Springer, Berlin, Heidelberg, May 1996. (Cited on 11, 15.)

[51] D. Pointcheval and G. Wang. VTBPEKE: Verifier-based two-basis password exponential key exchange. In R. Karri, O. Sinanoglu, A.-R. Sadeghi, and X. Yi, editors, *ASIACCS 17*, pages 301–312. ACM Press, Apr. 2017. (Cited on 2, 4, 28.)

[52] M. Rosulek. Is Cramer-Shoup backdoorable?, 2015. https://crypto.stackexchange.com/questions/27290/is-cramer-shoup-backdoorable. (Cited on 4, 23.)

[53] L. Rotem and G. Segev. Tighter security for schnorr identification and signatures: A high-moment forking lemma for $\Sigma$-protocols. In T. Malkin and C. Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 222–250, Virtual Event, Aug. 2021. Springer, Cham. (Cited on 4, 11, 12, 15.)

[54] C.-P. Schnorr. Efficient identification and signatures for smart cards. In G. Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, New York, Aug. 1990. (Cited on 11, 15.)

[55] C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, Jan. 1991. (Cited on 4.)

[56] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Rev.*, 41(2):303–332, 1999. (Cited on 3.)

[57] S. Tessaro and C. Zhu. Threshold and multi-signature schemes from linear hash functions. In C. Hazay and M. Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 628–658. Springer, Cham, Apr. 2023. (Cited on 16.)

## A Converses of signature theorems

In Section 3 we proved A-UF-CMA security of Okamoto (Theorem 3.1) and of Katz-Wang (Theorem 3.2) assuming standard UF-CMA security of Schnorr. Could A-UF-CMA of the former two schemes be proved with a weaker assumption? Here we prove that the answer is no; in fact A-UF-CMA security of Okamoto (and of Katz-Wang) is equivalent to UF-CMA security of Schnorr.

This is justified by proving the converses of Theorems 3.1 and 3.2. There, we gave theorem statements of the form: Given an adversary $B$ in game $\mathbf{G}^{\text{a-uf-cma}}_{\mathsf{OS},\mathsf{AGPg}_1^*,n}$ we could construct an adversary $A$ in game $\mathbf{G}^{\text{uf-cma}}_{\mathsf{SS},n}$ with similar advantage and running time. (Equivalently for the $\mathsf{KW}$ statement.) In the below theorems, we state and prove the converses. That is, given $A$ in game $\mathbf{G}^{\text{uf-cma}}_{\mathsf{SS},n}$ we can construct $B$ in game $\mathbf{G}^{\text{a-uf-cma}}_{\mathsf{OS},\mathsf{AGPg}_1^*,n}$ with similar advantage and running time. As in Section 3, the Okamoto result (Theorem A.1) holds for any hash function while the Katz-Wang result (Theorem A.2) assumes a random oracle.

```
Adversary $B(h, \mathbf{X}, \beta)$:
$(i, m, \sigma) \leftarrow A[\textsc{SignSim}, \textsc{HSim}](\varepsilon, \mathbf{X})$
$(e, y) \leftarrow \sigma$
$y_2 \leftarrow 1$
$y_1 \leftarrow (y - \beta \cdot y_2) \bmod p$
Return $(i, m, (e, y_1, y_2))$

Subroutine $\textsc{SignSim}(i, m)$:
$\sigma \leftarrow \mathbf{G}^{\text{a-uf-cma}}_{\mathsf{OS},\mathsf{AGPg}_1^*,n}.\textsc{Sign}(i, m)$
$(e, y_1, y_2) \leftarrow \sigma$
$y \leftarrow (y_1 + \beta \cdot y_2) \bmod p$
$\sigma' \leftarrow (e, y)$
Return $\sigma'$

Subroutine $\textsc{HSim}(z)$:
Return $\mathbf{G}^{\text{a-uf-cma}}_{\mathsf{OS},\mathsf{AGPg}_1^*,n}.\textsc{Hash}(z)$
```

Figure 19: Adversary $B$ for the proof of Theorem A.1.

**Theorem A.1** *Let $(\mathbb{G}, p, g)$ be a group specification. Let $\overline{\mathsf{HS}} \subseteq \mathsf{FUNC}(\mathbb{G} \times \mathbb{G} \times \{0,1\}^*, \mathbb{Z}_p)$ be a non-empty set of functions. Let $\mathsf{OS}$ and $\mathsf{SS}$ be the Okamoto and Schnorr signature schemes, respectively, associated to $(\mathbb{G}, p, g)$ and $\overline{\mathsf{HS}}$ as described in Section 3. Let $n > 0$ be a number of users. Given an adversary $A$ in game $\mathbf{G}^{\text{uf-cma}}_{\mathsf{SS},n}$ we can construct an adversary $B$ in game $\mathbf{G}^{\text{a-uf-cma}}_{\mathsf{OS},\mathsf{AGPg}_1^*,n}$ such that*

$$\mathbf{Adv}^{\text{uf-cma}}_{\mathsf{SS},n}(A) \leq \mathbf{Adv}^{\text{a-uf-cma}}_{\mathsf{OS},\mathsf{AGPg}_1^*,n}(B) \ . \tag{14}$$

*Adversary $B$ makes the same number of signing and hashing queries as $A$, and has running time close to that of $A$.*

**Proof of Theorem A.1:** We give adversary $B$ in Figure 19. $B$ runs adversary $A$ with access to the subroutines $\textsc{SignSim}, \textsc{HSim}$ as specified in the figure. On line 1, $A$ is initialized with empty public parameters and a list of verification keys $\mathbf{X}$ for the Schnorr scheme. These are of the form $\mathbf{X}[i] = g^{\mathbf{s}_1[i]} h^{\mathbf{s}_2[i]}$ from the Okamoto game. We may equivalently write this as $\mathbf{X}[i] = g^{\mathbf{s}[i]}$ where $\mathbf{s}[i] = (\mathbf{s}_1[i] + \beta \cdot \mathbf{s}_2[i]) \bmod p$. Thus adversary $B$ is setting up $A$'s perspective to be that of the Schnorr UF-CMA game with $\mathbf{s}$ as the list of secret keys. Moreover $\mathbf{X}$ is distributed correctly; we have $\mathbf{s}_1[i] \leftarrow\$ \mathbb{Z}_p$ from the Okamoto A-UF-CMA game which implies (from the relation above) that $\mathbf{s}[i]$ is also uniformly and independently random in $\mathbb{Z}_p$ for each $i$.

Next let us consider the $\textsc{SignSim}, \textsc{HSim}$ oracles. The latter simply forwards $A$'s $\textsc{HSim}$ queries to $B$'s own $\textsc{Hash}$ oracle. (From now on, we write $\textsc{Hash}$ to refer to the hash function both $A$ and $B$ are using.) The former requires some explanation. On lines 6-7 adversary $B$ forwards $A$'s query to its own signing oracle to receive an Okamoto signature $(e, y_1, y_2)$. As defined by the Okamoto scheme, we have $e = \textsc{Hash}(\mathbf{X}[i], g^{r_1} h^{r_2}, m)$ for nonces $r_1, r_2$ and we have for $j \in \{1, 2\}$, $y_j = (r_j + e\mathbf{s}_j[i]) \bmod p$. Now we claim that $\sigma' = (e, y)$ for $y = (y_1 + \beta \cdot y_2) \bmod p$ is a correct, and correctly distributed, Schnorr signature for the same user $i$ and message $m$. First, looking at $y$, we may write:

$$\begin{aligned} y = (y_1 + \beta \cdot y_2) \bmod p &= (r_1 + e\mathbf{s}_1[i]) + \beta \cdot (r_2 + e\mathbf{s}_2[i]) \bmod p \\ &= (r_1 + \beta r_2) + e(\mathbf{s}_1[i] + \beta \cdot \mathbf{s}_2[i]) \bmod p \\ &= (r_1 + \beta r_2) + e\mathbf{s}[i] \bmod p \ . \end{aligned}$$

The above is precisely a Schnorr signature with nonce $(r_1 + \beta r_2) \bmod p$. The nonce is correctly distributed; as long as $r_1 \leftarrow\$ \mathbb{Z}_p$ (as is done in Okamoto) we have $(r_1 + \beta r_2) \bmod p$ also uniformly distributed. Finally, this agrees with the $e$ value used in the signature. Recall that $e = \textsc{Hash}(\mathbf{X}[i], g^{r_1} h^{r_2}, m)$, which we can rewrite as $e = \textsc{Hash}(\mathbf{X}[i], g^{r}, m)$ where $r = (r_1 + \beta r_2) \bmod p$.

```
Adversary $B(h, \overline{\mathbf{X}}, \beta)$:
    For $i = 1 \ldots n$ do $(\mathbf{X}_g[i], \mathbf{X}_h[i]) \leftarrow \overline{\mathbf{X}}[i]$
    $(i, m, \sigma) \leftarrow A[\text{SignSim}, \text{HSim}](\varepsilon, \mathbf{X}_g)$
    Return $(i, m, \sigma)$

Subroutine $\text{SignSim}(i, m)$:
    $\sigma \leftarrow \mathbf{G}^{\text{a-uf-cma}}_{\mathsf{KW},\mathsf{AGPg}^*_1,n}.\text{Sign}(i, m)$
    Return $\sigma$

Subroutine $\text{HSim}(z)$:
    If $\text{HT}[z] \neq \bot$ then return $\text{HT}[z]$
    $(a, b, m) \leftarrow z$ // Parse $z$ as shown where $a, b \in \mathbb{G}$ and $m \in \{0,1\}^*$
    $\text{HT}[z] \leftarrow \mathbf{G}^{\text{a-uf-cma}}_{\mathsf{KW},\mathsf{AGPg}^*_1,n}.\text{Hash}((a, a^\beta), b, b^\beta, m)$
    Else if parsing fails $\text{HT}[z] \leftarrow_\$ \mathbb{Z}_p$
    Return $\text{HT}[z]$
```

Figure 20: Adversary $B$ for the proof of Theorem A.2.

This is exactly what is hashed for a Schnorr signature with nonce $r$.

Thus signature queries are correctly answered. We lastly argue that the forgery output by $B$ on line 5 wins the A-UF-CMA game as long as $A$'s forgery on line 1 wins its UF-CMA game. First, since $i, m$ are unchanged, $B$'s forgery is fresh as long as $A$'s is. As for correctness, if $(e, y)$ is a correct Schnorr signature we have $e = \text{Hash}(\mathbf{X}[i], g^y \mathbf{X}[i]^{-e}, m)$. Defining $y_1, y_2$ as we do on lines 3,4 we then have

$$\begin{aligned} e &= \text{Hash}(\mathbf{X}[i], g^y \mathbf{X}[i]^{-e}, m) \\ &= \text{Hash}(\mathbf{X}[i], g^{y_1+\beta y_2} \mathbf{X}[i]^{-e}, m) \\ &= \text{Hash}(\mathbf{X}[i], g^{y_1} h^{y_2} \mathbf{X}[i]^{-e}, m) \ . \end{aligned}$$

This last line matches verification of the Okamoto scheme, as per lines 13,14 of Figure 5. Note that we select $y_2 = 1$ for concreteness, but the argument holds for any $y_2 \in \mathbb{Z}_p$. We conclude the proof by remarking that $B$ makes the same number of signing and hashing queries as $A$, and otherwise has running time close to that of $A$. ▌

**Theorem A.2** *Let $(\mathbb{G}, p, g)$ be a group specification. Let* KW *be the associated Katz-Wang signature scheme as described in Section 3. Let* SS *be the Schnorr signature scheme associated to $(\mathbb{G}, p, g)$ and $\mathsf{FUNC}(\mathbb{G} \times \mathbb{G} \times \{0,1\}^*, \mathbb{Z}_p)$. Let $n > 0$ be a number of users. Given an adversary $A$ in game $\mathbf{G}^{\text{uf-cma}}_{\mathsf{SS},n}$ we can construct an adversary $B$ in game $\mathbf{G}^{\text{a-uf-cma}}_{\mathsf{KW},\mathsf{AGPg}^*_1,n}$ such that*

$$\mathbf{Adv}^{\text{uf-cma}}_{\mathsf{SS},n}(A) \leq \mathbf{Adv}^{\text{a-uf-cma}}_{\mathsf{KW},\mathsf{AGPg}^*_1,n}(B) \ . \qquad (15)$$

*Adversary $B$ makes at most the same number of signing and hashing queries as $A$, and has running time close to that of $A$.*

**Proof of Theorem A.2:** We give adversary $B$ in Figure 20. In the Katz-Wang A-UF-CMA game, $B$ receives as input $h, \beta$ and verification keys $\overline{\mathbf{X}}$ where $\overline{\mathbf{X}}[i]$ is a tuple that we write as $(\mathbf{X}_g[i], \mathbf{X}_h[i])$. Recall that Katz-Wang selects verification keys such that, for secret keys $\mathbf{s}$, we have

$\mathbf{X}_g[i] = g^{\mathbf{s}[i]}$ and $\mathbf{X}_h[i] = \mathbf{X}_g[i]^\beta$. Thus $\mathbf{X}_g$, as given to adversary $A$ on line 2, is a correct list of Schnorr verification keys corresponding to the same secret keys $\mathbf{s}$.

Next let us consider the SignSim, HSim oracles. The former simply forwards signing queries to its own oracle and returns them, unchanged. In HSim we program the random oracle to ensure that these signature responses are correct. In particular we have

$$\text{HSim}(a, b, m) = \mathbf{G}^{\text{a-uf-cma}}_{\mathsf{KW},\mathsf{AGPg}^*_1,n}.\text{Hash}((a, a^\beta), b, b^\beta, m) ,$$

as implemented on lines 7-8 and assuming parsing works. This implies, then, that for the verification keys at hand, and for any exponent $r \in \mathbb{Z}_p$, we get

$$\begin{aligned}\text{HSim}(\mathbf{X}_g[i], g^r, m) &= \mathbf{G}^{\text{a-uf-cma}}_{\mathsf{KW},\mathsf{AGPg}^*_1,n}.\text{Hash}((\mathbf{X}_g[i], \mathbf{X}_g[i]^\beta), g^r, g^{r\beta}, m) \\ &= \mathbf{G}^{\text{a-uf-cma}}_{\mathsf{KW},\mathsf{AGPg}^*_1,n}.\text{Hash}((\mathbf{X}_g[i], \mathbf{X}_h[i]), g^r, h^r, m) .\end{aligned}$$

On this most recent line, the righthand expression is the $e$ value computed for a Katz-Wang signature on message $m$ using nonce $r$. On the prior line, the lefthand expression is the $e$ value computed for a Schnorr signature, also on message $m$ and nonce $r$. Hence we may use $e$ as a correct signature component for both schemes. Moreover, $y = (r + e\mathbf{s}[i]) \bmod p$ is computed identically in Katz-Wang and in Schnorr. All of this proves that the $\sigma = (e, y)$ values returned by SignSim are in fact correct Schnorr signatures, given that they are selected as correct Katz-Wang signatures.

Adversary $B$ finally outputs the same forgery $(i, m, \sigma)$ as $A$ on line 3. As argued above, this is a correct Katz-Wang signature, under $\mathbf{G}^{\text{a-uf-cma}}_{\mathsf{KW},\mathsf{AGPg}^*_1,n}.\text{Hash}$, if it is a correct Schnorr signature under HSim. We conclude by noting that $B$ makes at most the same number of signing and hashing queries as $A$, and has running time close to that of $A$. The same set of queried $(i, m)$ is tracked in both games, so $B$'s forgery is fresh as long as $A$'s is. ▌

## B Multi-user DT-DDH is implied by single-user

In Section 4 we proved A-IND-CCA1 security of Cramer-Shoup assuming DT-DDH, where both games were in the $n$-user setting. The latter game $\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}$ was defined in Figure 12. In this section we prove that security in game $\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}$ is asymptotically equivalent to security in the single-user case $\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,1}$. This is known, with a tight reduction, for standard DDH [14]. As DT-DDH is a non-standard assumption, we prove its corresponding reduction here, albeit non-tightly, with a factor of $n$ in the advantage bound. The challenge in removing this factor is that game $\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,1}$ allows for only a single GetTgt query, after which *all* DH queries are disallowed; it is not obvious how an adversary $A_1$ could answer an adversary $A_n$ who intersperses $n$ GetTgt queries among DH queries to any of the $n$ users.

Below, we formally state and prove our result, that multi-user security of DT-DDH is implied by single-user. (The opposite direction is proved immediately.) The proof uses a standard hybrid argument.

**Theorem B.1** *Let $(\mathbb{G}, p, g)$ be a group specification and $n > 0$ a number of users. Given an adversary $A_n$ in game $\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}$ we can construct an adversary $A_1$ in game $\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,1}$ such that*

$$\mathbf{Adv}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A_n) \le n \cdot \mathbf{Adv}^{\text{dt-ddh}}_{\mathbb{G},p,g,1}(A_1) . \quad (16)$$

*Adversary $A_1$ makes at most the same number of* DH *queries as $A_n$, at most one* GetTgt *query, and has running time close to that of $A_n$.*

```
Games $G_0, \dots, G_J, \dots, G_n$

INIT:
1  For $i = 1, \dots, n$ do $\mathbf{z}[i] \leftarrow\!\$\ \mathbb{Z}_p$ ; $\mathbf{Z}[i] \leftarrow g^{\mathbf{z}[i]}$
2  For $i = 1, \dots, n$ do $\mathbf{T}[i] \leftarrow\!\$\ \mathbb{G}$ ; $\mathbf{R}[i] \leftarrow\!\$\ \mathbb{G}$
3  For $i = 1, \dots, J$ do $\mathbf{S}[i] \leftarrow \mathbf{T}[i]^{\mathbf{z}[i]}$
4  For $i = (J+1), \dots, n$ do $\mathbf{S}[i] \leftarrow \mathbf{R}[i]$
5  Return $\mathbf{Z}$

DH$(i, Y)$:
6  If done$[i]$ then return $\perp$
7  Return $Y^{\mathbf{z}[i]}$

GETTGT$(i)$:
8  done$[i] \leftarrow$ true
9  Return $(\mathbf{T}[i], \mathbf{S}[i])$

FIN$(d')$:
10 Return $d'$
```

```
Adversary $B_J(Z^*)$:
1  For $i = 1, \dots, n$ do
2      If $(i = J)$ then $\mathbf{Z}[J] \leftarrow Z^*$
3      Else
4          $\mathbf{z}[i] \leftarrow\!\$\ \mathbb{Z}_p$ ; $\mathbf{Z}[i] \leftarrow g^{\mathbf{z}[i]}$
5          $\mathbf{T}[i] \leftarrow\!\$\ \mathbb{G}$ ; $\mathbf{R}[i] \leftarrow\!\$\ \mathbb{G}$
6  For $i = 1, \dots, (J-1)$ do $\mathbf{S}[i] \leftarrow \mathbf{T}[i]^{\mathbf{z}[i]}$
7  For $i = (J+1), \dots, n$ do $\mathbf{S}[i] \leftarrow \mathbf{R}[i]$
8  $d' \leftarrow A_n[\text{DHSIM}, \text{GETTGTSIM}](\mathbf{Z})$
9  Return $d'$

Subroutine DHSIM$(i, Y)$:
10 If done$[i]$ then return $\perp$
11 If $(i = J)$ then return DH$(1, Y)$
12 Else return $Y^{\mathbf{z}[i]}$

Subroutine GETTGTSIM$(i)$:
13 done$[i] \leftarrow$ true
14 If $(i = J)$ then $(T^*, S^*) \leftarrow$ GETTGT$(1)$ ; return $(T^*, S^*)$
15 Else return $(\mathbf{T}[i], \mathbf{S}[i])$
```

Figure 21: **Left:** Games $G_J$ for the proof of Theorem B.1, for $0 \le J \le n$. **Right:** Adversaries $B_J$ for the proof of Theorem B.1, for $1 \le J \le n$.

---

**Proof of Theorem B.1:** This proof refers to the games and adversaries in Figure 21. There, over index $J$, we define games $G_J$ for $0 \le J \le n$ and we give adversaries $B_J$ for $1 \le J \le n$. We begin by recalling the definition of DT-DDH advantage, such that for challenge bit $d$, we can write

$$\begin{aligned}\mathbf{Adv}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A_n) &= 2 \cdot \Pr[\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A_n)] - 1 \\ &= \Pr[\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A_n) \mid d = 1] + \Pr[\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A_n) \mid d = 0] - 1 \; .\end{aligned}$$

Next, we consider $G_J$ for $J = n$ and $J = 0$. In general, game $G_J$ selects "real" DT-DDH values for $\mathbf{S}[1], \dots, \mathbf{S}[J]$ (on line 3) and "random" DT-DDH values for $\mathbf{S}[J+1], \dots, \mathbf{S}[n]$ (on line 4). Otherwise the DH and GETTGT oracles are as usual in the DT-DDH game. (For now, ignore FIN.) Now, game $G_n$ is exactly game $\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}$ with challenge bit $d = 1$ because all $n$ users have a "real" challenge value $\mathbf{S}[i] = \mathbf{T}[i]^{\mathbf{z}[i]}$. On the opposite end, game $G_0$ is exactly game $\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}$ with challenge bit $d = 0$ because all of the challenge values are "random" with $\mathbf{S}[i] = \mathbf{R}[i]$. Thus, we argue that $\Pr[\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A_n) \mid d = 1]$ is the probability that $A_n$ outputs $d' = 1$ in game $G_n$. Similarly, $\Pr[\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A_n) \mid d = 0]$ is the probability that $A_n$ outputs $d' = 0$ in game $G_0$. Hence, because game $G_J$ in FIN simply returns the guess $d'$ (and is true iff $d' = 1$), we can write

$$\begin{aligned}\mathbf{Adv}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A_n) &= \Pr[\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A_n) \mid d = 1] + \Pr[\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A_n) \mid d = 0] - 1 \\ &= \Pr[\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A_n) \mid d = 1] - \Big(1 - \Pr[\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A_n) \mid d = 0]\Big) \\ &= \Pr[G_n(A_n)] - \Pr[G_0(A_n)] \; .\end{aligned}$$

Moving on to general $J$, we claim that for all $1 \le J \le n$, we have

$$\Pr[\mathrm{G}_J(A_n)] - \Pr[\mathrm{G}_{J-1}(A_n)] \le \mathbf{Adv}^{\text{dt-ddh}}_{\mathbb{G},p,g,1}(B_J)\ . \tag{17}$$

Let us explain why the theorem statement follows from Eq. (17). From above, we have

$$\begin{aligned}\mathbf{Adv}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A_n) &= \Pr[\mathrm{G}_n(A_n)] - \Pr[\mathrm{G}_0(A_n)] \\ &= \Pr[\mathrm{G}_n(A_n)] - \Pr[\mathrm{G}_0(A_n)] + \sum_{J=1}^{n-1} \Pr[\mathrm{G}_J(A_n)] - \sum_{J=1}^{n-1} \Pr[\mathrm{G}_J(A_n)] \\ &= \sum_{J=1}^{n} \Pr[\mathrm{G}_J(A_n)] - \Pr[\mathrm{G}_{J-1}(A_n)] \\ &\le \sum_{J=1}^{n} \mathbf{Adv}^{\text{dt-ddh}}_{\mathbb{G},p,g,1}(B_J)\ .\end{aligned}$$

Finally, consider the adversary $A_1$ which selects at random $J \leftarrow\$ [1..n]$ and then operates as $B_J$. The advantage of $A_1$ in game $\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,1}$ is

$$\mathbf{Adv}^{\text{dt-ddh}}_{\mathbb{G},p,g,1}(A_1) = \frac{1}{n} \cdot \sum_{J=1}^{n} \mathbf{Adv}^{\text{dt-ddh}}_{\mathbb{G},p,g,1}(B_J)$$

which we can combine with the above to obtain

$$\mathbf{Adv}^{\text{dt-ddh}}_{\mathbb{G},p,g,n}(A_n) \le n \cdot \mathbf{Adv}^{\text{dt-ddh}}_{\mathbb{G},p,g,1}(A_1)\ ,$$

matching the theorem statement. Adversary $A_1$ in all cases makes one GETTGT query and at most the same number of DH queries as $A_n$, as needed.

Lastly, it remains to justify Eq. (17). To do so, observe that games $\mathrm{G}_J$ and $\mathrm{G}_{J-1}$ differ only in their selection of $\mathbf{S}[J]$. While $\mathrm{G}_J$ selects $\mathbf{S}[J] = \mathbf{T}[J]^{\mathbf{z}[J]}$, game $\mathrm{G}_{J-1}$ instead selects $\mathbf{S}[J] = \mathbf{R}[J]$. All other indices of $\mathbf{S}$ match. That is, all indices prior to $J$ are "real" and all indices after $J$ are "random" in both games. Now consider adversary $B_J$ in game $\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,1}$. At a high level, $B_J$ embeds its single DT-DDH challenge $(Z^*, T^*, S^*)$ at user $J$. This happens at line 2, line 11 and line 14 of the adversary in Figure 21. For all other users, $B_J$ selects the game values on its own, in particular so that all indices prior to $J$ are "real" and all indices after $J$ are "random."

Now, if $B_J$ is in game $\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,1}$ with challenge bit $d = 1$, this means that $(Z^*, T^*, S^*)$ is a "real" tuple and thus user $J$ matches game $\mathrm{G}_J$. If $B_J$ is in game $\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,1}$ with challenge bit $d = 0$, then $(Z^*, T^*, S^*)$ is a "random" tuple and user $J$ instead matches game $\mathrm{G}_{J-1}$. On line 9, $B_j$ outputs the same guess $d'$ as $A_n$. In particular,

$$\begin{aligned}\mathbf{Adv}^{\text{dt-ddh}}_{\mathbb{G},p,g,1}(B_J) &= 2 \cdot \Pr[\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,1}(B_J)] - 1 \\ &= \Pr[\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,1}(B_J) \mid d = 1] + \Pr[\mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,1}(B_J) \mid d = 0] - 1 \\ &= \Pr[B_J \text{ guesses 1 in } \mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,1} \mid d = 1] - \Pr[B_J \text{ guesses 1 in } \mathbf{G}^{\text{dt-ddh}}_{\mathbb{G},p,g,1} \mid d = 0] \\ &\ge \Pr[A_n \text{ guesses 1 in } \mathrm{G}_J] - \Pr[A_n \text{ guesses 1 in } \mathrm{G}_{J-1}] \\ &= \Pr[\mathrm{G}_J(A_n)] - \Pr[\mathrm{G}_{J-1}(A_n)]\ .\end{aligned}$$

The last line is precisely Eq. (17). This completes the proof. ▌

## C Proof of Theorem 5.3

**Proof of Theorem 5.3:** The proof closely follows the one of Abdalla and Barbosa [2]. Nonetheless, we are considering $A$ in game $\mathbf{G}^{\text{a-pake}}_{\mathsf{SPAKE2},\mathsf{AGPg}_2}$ rather than $\mathbf{G}^{\text{pake}}_{\mathsf{SPAKE2}}$ and are ultimately proving a different advantage bound. We use the sequence of games shown in Figures 22 and 23.

<u>Game $G_0$.</u> This is the original A-PAKE security game, parameterized by advice algorithm $\mathsf{AGPg}_2$ since SPAKE2 has two group-element parameters $M, N$. We have

$$\mathbf{Adv}^{\text{a-pake}}_{\mathsf{SPAKE2},\mathsf{AGPg}_2}(A) = 2\Pr[\mathrm{G}_0(A)] - 1 \ .$$

Note that line 1 selects random group elements, not generators, to implement $\mathsf{AGPg}_2$. Also, instead of the game selecting a hash function $\mathsf{H} \leftarrow_\$ \mathsf{SPAKE2.HS}$, the oracle Hash in game $G_0$ implements a random mapping from $\{0,1\}^*$ to $\mathcal{K}$, which is equivalent.

<u>Game $G_1$.</u> In this game, we return $\perp$ when two traces collide (among parties that should not be considered partnered) in any of the Execute or Send oracles. This is implemented on lines 8-9 for the Execute oracle, on lines 38-39 for the SendResp oracle and on lines 48-49 for the SendTermInit oracle. Note that SendInit does not yet involve a full trace, so we there omit a change. Also, in the SendTermInit oracle, we include an exception (the $\setminus\{\mathsf{S}\}$ component) for a server partnered to the current client, as these parties are expected to have the same trace. For the first two locations (Execute and SendResp), it is evident from the preceding lines that the traces only collide if one (or two) uniformly random $X^*, Y^*$ collide. This can be bounded by the birthday bound.

We claim the same for the third location (SendTermInit) but this requires care. First, the SendTermInit oracle always returns $\perp$, regardless of the check on line 48, so the return value is unchanged. However, if $\perp$ is returned on line 49 rather than on line 56, then the instance $\tau^i_{\mathsf{C}}$ will not be updated to include $Y^*$ and $\mathsf{SK}$, which could cause a difference from $G_0$ in later queries. But, note that games $G_0, G_1$ both include line 46 which first asserts, before proceeding in the oracle, that $\tau^i_{\mathsf{C}} = (x, (\mathsf{C}, \mathsf{S}, X^*, \perp), \perp, \mathsf{false})$. In particular, $X^*$ is populated in the state already but $Y^*$ is not. Such a scenario only arises from a SendInit query, which selects $X^*$ as a random group element on line 33. Hence, while $Y^*$ is selected by the adversary, the $X^*$ value involved in a passing line 48 must be a uniformly random group element.

In total, then, over the three locations where game $G_1$ returns $\perp$ if traces collide, this check is always over one (or two) uniformly random group element(s). Thus the probability that traces collide is bounded by the birthday bound, and we can bound the difference between games $G_0, G_1$ by

$$|\Pr[\mathrm{G}_1(A)] - \Pr[\mathrm{G}_0(A)]| \leq \frac{(q_{\mathrm{s}} + q_{\mathrm{e}})^2}{p} \ .$$

<u>Game $G_2$.</u> In this game, we make the freshness condition explicit by adding a variable $\mathsf{fr}$ to each instance. Now an instance $\tau^i_{\mathsf{P}}$ is a tuple $(\mathsf{e}, \mathsf{tr}, \mathsf{sk}, \mathsf{acc}, \mathsf{fr})$. This addition is shown in the boxed code in Figure 22. This is only a conceptual change, hence

$$\Pr[\mathrm{G}_2(A)] = \Pr[\mathrm{G}_1(A)] \ .$$

Note that in Figure 22, we use the notation $\mathsf{fr} \leftarrow [\![ \mathit{expr} ]\!]$ to evaluate the boolean expression *expr* and assign the result to $\mathsf{fr}$.

With freshness defined as in Section 5.1, let us consider the additions to game $\mathrm{G}_2$. First, in Execute queries, both client and server are labeled as $\mathsf{fr} = \mathsf{true}$ on lines 15,16 as per condition (3.1). In a Reveal$(\mathsf{P}, i)$ query, the instance $\tau_{\mathsf{P}}^{i}$ itself, and any partners, are labeled as $\mathsf{fr} = \mathsf{false}$. Since we have already required unique traces, there is at most one partner, and we do not violate (3.2) here. The Test oracle simply uses the variable $\mathsf{fr}$ rather than the function $\mathsf{Fresh}$. Moving on to the Send oracles, we set $\mathsf{fr} = \mathsf{false}$ in SendInit because the instance has not accepted yet. In SendResp, the instance $\tau_{\mathsf{S}}^{i}$ cannot yet have a partner because the trace is unfinished. It's also not part of an Execute query because of the requirement on line 36. Line 40 thus implements the only relevant freshness check, which is that there is no parter and Corrupt was not queried. Finally, consider SendTermInit. Instance $\tau_{\mathsf{C}}^{i}$ could either have one or zero partners; it cannot have two (a potential partner $\tau_{\mathsf{S}}^{j}$ could only be defined during SendResp or Execute, which both return $\perp$ if a trace collides with a prior instance). Now, if $\tau_{\mathsf{C}}^{i}$ has zero partners, condition (3.4) is implemented on line 51. Else, if it has one partner, condition (3.3) is implemented on line 50. There, we use "$\exists!$" to make clear that the check is for exactly one partner, but as argued above, there is at most one $j$ that could pass line 50.

Game $\mathrm{G}_3$. In this game, we pick session keys for Execute queries uniformly at random from the key space. In particular, this is specified on line 14 for $\mathrm{G}_2$ versus line 12 for $\mathrm{G}_3$. To bound the difference between $\mathrm{G}_2$ and $\mathrm{G}_3$, we construct an adversary $B$ against the strong computational Diffie-Hellman problem (SCDH). The description of $B$ is given in Figure 24. Note that this is specifically in the $q_{\mathrm{e}}$-user strong CDH game, but this is tightly equivalent to the single-user version. We claim:

$$|\Pr[\mathrm{G}_3(A)] - \Pr[\mathrm{G}_2(A)]| \leq \Pr[\mathrm{G}_3(A) \text{ sets } \mathsf{bad}] \tag{18}$$

$$\Pr[\mathrm{G}_3(A) \text{ sets } \mathsf{bad}] \leq \mathbf{Adv}^{\mathrm{scdh}}_{\mathbb{G},g,p,q_{\mathrm{e}}}(B) \ . \tag{19}$$

To explain Eq. (18), we observe that game $\mathrm{G}_3$ only differs from $\mathrm{G}_2$ if it encounters a query in contradiction with its selection of $\mathsf{SK}$ values. (Else, it is expected that Hash selects them at random from $\mathcal{K}$, as game $\mathrm{G}_3$ may do on line 12 instead.) We let the table HT keep track of Hash responses, in particular mapping instance information $(\mathsf{C}, \mathsf{S}, X^*, Y^*, \mathsf{pw}_{\mathsf{CS}}, g^{xy})$ to the associated session key $\mathsf{SK}$. We let table $\mathrm{T}_{\mathrm{e}}$ contain the same information for keys assigned during an Execute query in $\mathrm{G}_3$. Now flag $\mathsf{bad}$ is set on line 11 if Execute encounters an instance which already has an assigned key in HT, or $\mathsf{bad}$ is set on line 27 if Hash encounters an instance which already has an assigned key in $\mathrm{T}_{\mathrm{e}}$. Otherwise $\mathrm{G}_2, \mathrm{G}_3$ are identical-until-$\mathsf{bad}$, which proves Eq. (18).

To prove Eq. (19), the reduction again follows [2]. We give adversary $B$ in Figure 24. Adversary $B$ runs $A$ with access to subroutines InitSim, ExecuteSim and HSim as shown in Figure 24, while the remaining oracles are implemented according to game $\mathrm{G}_3$. Adversary $B$ may output its SCDH answer during either an ExecuteSim (line 13) or HSim (line 23) response, or will output $\perp$ if $A$ terminates without having triggered either of these lines.

Recall that in the $q_{\mathrm{e}}$-user SCDH game, $B$ receives as input $X_1, \ldots, X_{q_{\mathrm{e}}}, Y_1, \ldots, Y_{q_{\mathrm{e}}}$ and is attempting to output some $(Y_i)^{x_i}$ value. We claim that if $\mathrm{G}_3(A)$ sets $\mathsf{bad}$ on line 11 then $B$ will output a winning SCDH answer on line 13; and if $\mathrm{G}_3(A)$ sets $\mathsf{bad}$ on line 27 then $B$ will output a winning SCDH answer on line 23 (here, line numbers refer to the respective $\mathrm{G}_3$ and $B$ figures).

Let us start by explaining ExecuteSim queries. In the $k$-th ExecuteSim query made by $A$, $B$ embeds its $k$-th SCDH challenge $X_k, Y_k$. Specifically, $B$ sets the protocol execution to consist of $X^* = X_k \cdot g^{m \cdot \mathsf{pw}_{\mathsf{CS}}}$ and $Y^* = Y_k \cdot g^{n \cdot \mathsf{pw}_{\mathsf{CS}}}$. Since $X_k = g^{x_k}$ for $x_k \leftarrow_{\$} \mathbb{Z}_p$ in the SCDH game (and $Y_k$ similarly), this selection of $X^*, Y^*$ matches the SPAKE2 specification. The remainder of

ExecuteSim (ignoring lines 11-13 momentarily) matches game $G_3$. Since $B$ does not know $Y_k^{x_k}$ at this point, it is not included in the $T_e$ information; however $Y_k^{x_k}$ is completely determined by $X^*, Y^*, \mathsf{pw}_{\mathsf{CS}}$. Now consider the table $T_{cdh}$ used on lines 11-13. Looking ahead, this is populated on line 24 only if a DDH query returns true, which in particular checks that for some $k'$ where $X_{k'} = g^{x_{k'}}$ and $(Y^*/N^{\mathsf{pw}_{\mathsf{CS}}}) = Y_{k'}$, we have $Y_{k'}^{x_{k'}} = Z$. In fact this holds for the current round $k$, which uses the same $X^*, Y^*, \mathsf{pw}_{\mathsf{CS}}$. Thus the $Z$ finalized on line 13 is such that $Y_k^{x_k} = Z$, which makes $(k, Z)$ a winning SCDH output. When does line 11 pass? Only if $(\mathsf{C}, \mathsf{S}, X^*, Y^*, \mathsf{pw}_{\mathsf{CS}}, Y_k^{x_k})$ has been encountered in a Hash query already, which is exactly the condition that sets bad on lines 10-11 of $G_3$.

Next let us consider Hash queries, as implemented by HSim. Lines 19 and 25-27 are as usual. Line 20 checks whether the password in the query is correct, and line 21 checks whether the query corresponds to an SCDH input pair $(X_{k'}, Y_{k'})$, as would be embedded in an Execute query. If there is a match, line 22 then checks to see if the $Z$ component of the Hash input is in fact $Y_{k'}^{x_{k'}}$. If it is, it is either returned as an SCDH answer on line 23, or is saved on line 24 to be returned by a future Execute query. (In either case, requiring that the DDH oracle result be true ensures that the returned $(k', Z)$ is correct.) When does line 23 pass? It is exactly when Hash is queried on an instance (consisting of the given $X^*, Y^*, \mathsf{pw}_{\mathsf{CS}}$) that already appeared in an Execute query. This is the same condition that sets bad on line 27 of $G_3$.

The analysis of these two cases proves Eq. (19); namely $B$ wins its SCDH game whenever $G_3(A)$ sets bad. Finally note that $B$ makes at most $q_h$ queries to its DDH oracle and otherwise has running time about that of $A$.

Game $G_4$. The next game is given in the following figure, Figure 23. Here, we remove the lines that are no longer needed, and indicate with boxes the lines that differ from $G_3$. We claim that these differing lines do not actually change the operation of the game; they merely make obvious that messages $X^*, Y^*$ can be equivalently selected without depending on the password $\mathsf{pw}_{\mathsf{CS}}$. To explain, we note that the instruction "$x \leftarrow\$ \mathbb{Z}_p$ ; $X^* \leftarrow g^x \cdot g^{m \cdot \mathsf{pw}_{\mathsf{CS}}}$" is equivalent to "$\bar{x} \leftarrow\$ \mathbb{Z}_p$ ; $X^* \leftarrow g^{\bar{x}}$ ; $x \leftarrow \bar{x} - m \cdot \mathsf{pw}_{\mathsf{CS}}$" and similarly for $Y^*$. Hence, the game proceeds identically to the previous one, and we have

$$\Pr[G_4(A)] = \Pr[G_3(A)] .$$

Game $G_5$. In this final game, we modify the Send and Hash oracles. In brief, we aim to have game $G_5$ always select session keys and protocol messages uniformly at random, so that, crucially, they are independent of passwords. (Game $G_3$ already did this for Execute queries, and $G_4$ did this for protocol messages $X^*, Y^*$.) The consequence of this is that no oracles provide any information about passwords, other than the Hash oracle. We claim that we can bound the difference between games $G_4, G_5$ as:

$$|\Pr[G_5(A)] - \Pr[G_4(A)]| \leq \frac{q_h}{|\mathcal{PW}|} . \tag{20}$$

Let us proceed to the details, starting with the Send oracles. In SendResp, game $G_4$ selects the session key by hashing on line 55. Game $G_5$ alters this selection on lines 48-54, only if the instance is fresh. A difference could only arise on lines 51-54. Here we set bad if a session key for this instance is already determined by a prior Hash response, on lines 51-52. On line 53 we select $\mathsf{SK} \leftarrow\$ \mathcal{K}$ and then store this information in table $T_s$. This does not change the oracle response but this stored information could cause the game to set bad in a later Hash query; this occurs on lines 27-29 of Hash.

Thus for SendResp there are two places where $\mathsf{bad}$ could be set, indicating a difference between $\mathrm{G}_4, \mathrm{G}_5$. In brief, it is when a session key $\mathsf{SK}$ selected in SendResp disagrees with the output of the Hash oracle, or vice versa, when Hash is queried on the appropriate instance inputs. Oracle SendTermInit is modified in a similar way as SendResp in $\mathrm{G}_5$. On lines 68-69, the game checks if $\mathsf{SK}$ has already been determined during a SendResp response, in which case it continues on with that key. On lines 71-74 the game again selects $\mathsf{SK}$ uniformly at random, checking for disagreement with the Hash oracle and storing the information in table $\mathrm{T}_\mathrm{s}$. This information is checked again for disagreement on lines 30-32 of the Hash oracle.

Now, what is the probability that $\mathsf{bad}$ is set during execution of game $\mathrm{G}_5$? Observe that in all locations where $\mathsf{bad}$ is set, the correct password $\mathsf{pw}_{\mathsf{CS}}$ must have been queried, along with the rest of the instance information, to the Hash oracle. Either this is because the game checks that the password is $\mathsf{pw}_{\mathsf{CS}}$ (lines 28,31), or the stored hash input must contain $\mathsf{pw}_{\mathsf{CS}}$ (lines 51,71). However, all of the game responses (aside from Hash) are independent of, and thus provide no information about, $\mathsf{pw}_{\mathsf{CS}}$. Recall that this was chosen via $\mathsf{pw}_{\mathsf{CS}} \leftarrow\$\, \mathcal{PW}$ on line 3. Now $\mathsf{bad}$ can only be set if the adversary is able to guess a correct $\mathsf{pw}_{\mathsf{CS}}$ to include in a Hash query. Over $q_\mathrm{h}$ queries, and assuming uniformly random passwords, this is at most $q_\mathrm{h}/|\mathcal{PW}|$. This completes the justification of Eq. (20).

Finally, we claim

$$\Pr[\mathrm{G}_5(A)] = \frac{1}{2} \ .$$

This is because in $\mathrm{G}_5$, the session key is always selected as $\mathsf{SK} \leftarrow\$\, \mathcal{K}$. Recall that the A-PAKE game's challenge bit $d$ determines whether, for a fresh instance in a Test query, a true session key $\mathsf{SK}_0$ or a random key $\mathsf{SK}_1$ is returned. (This occurs on lines 23,24 of $\mathrm{G}_5$.) When the true session keys are also chosen as $\mathsf{SK}_0 \leftarrow\$\, \mathcal{K}$, an adversary $A$ has no advantage in distinguishing between $\mathsf{SK}_0, \mathsf{SK}_1$, meaning that the probability of guessing $d$ is $1/2$. To conclude, collecting the terms from the above sequence of game hops, we prove the equation in the theorem statement. ▌

```
Games $\mathrm{G}_0 = \mathbf{G}^{\text{a-pake}}_{\mathsf{SPAKE2},\mathsf{AGPg}_2}$, $\mathrm{G}_1$, $\boxed{\mathrm{G}_2, \mathrm{G}_3}$

INIT:
$d \leftarrow_{\$} \{0,1\}$ ; $m, n \leftarrow_{\$} \mathbb{Z}_p$ ; $M \leftarrow g^m$ ; $N \leftarrow g^n$
$\pi \leftarrow (M, N)$ ; $ad \leftarrow (m, n)$
For $(\mathsf{C},\mathsf{S}) \in \mathcal{C} \times \mathcal{S}$ do $\mathsf{pw}_{\mathsf{CS}} \leftarrow_{\$} \mathcal{PW}$
Return $(\pi, ad)$

EXECUTE($\mathsf{C}, i, \mathsf{S}, j$):
Require: $\tau^i_{\mathsf{C}} = \bot \wedge \tau^j_{\mathsf{S}} = \bot$
$x \leftarrow_{\$} \mathbb{Z}_p$ ; $X^* \leftarrow g^x \cdot g^{m\cdot \mathsf{pw}_{\mathsf{CS}}}$
$y \leftarrow_{\$} \mathbb{Z}_p$ ; $Y^* \leftarrow g^y \cdot g^{n\cdot \mathsf{pw}_{\mathsf{CS}}}$
If $\exists\, \mathsf{P} \in \mathcal{C} \cup \mathcal{S}$ and $j'$ s.t. $\tau^{j'}_{\mathsf{P}}.\mathsf{tr} = (\mathsf{C},\mathsf{S},X^*,Y^*)$ then      // $\mathrm{G}_1$-$\mathrm{G}_3$
    Return $\bot$                                                       // $\mathrm{G}_1$-$\mathrm{G}_3$
If $\mathrm{HT}[\mathsf{C},\mathsf{S},X^*,Y^*,\mathsf{pw}_{\mathsf{CS}},g^{xy}] \neq \bot$ then        // $\mathrm{G}_3$
    $\mathsf{bad} \leftarrow \mathsf{true}$ ; Return $\bot$                                  // $\mathrm{G}_3$
$\mathsf{SK} \leftarrow_{\$} \mathcal{K}$                                                     // $\mathrm{G}_3$
$\mathrm{T_e}[\mathsf{C},\mathsf{S},X^*,Y^*,\mathsf{pw}_{\mathsf{CS}},g^{xy}] \leftarrow \mathsf{SK}$                 // $\mathrm{G}_3$
$\mathsf{SK} \leftarrow \mathrm{HASH}(\mathsf{C},\mathsf{S},X^*,Y^*,\mathsf{pw}_{\mathsf{CS}},X^y)$              // $\mathrm{G}_0$-$\mathrm{G}_2$
$\tau^i_{\mathsf{C}} \leftarrow (x, (\mathsf{C},\mathsf{S},X^*,Y^*), \mathsf{SK}, \mathsf{true}, \boxed{\mathsf{true}})$
$\tau^j_{\mathsf{S}} \leftarrow (y, (\mathsf{C},\mathsf{S},X^*,Y^*), \mathsf{SK}, \mathsf{true}, \boxed{\mathsf{true}})$
Return $(X^*, Y^*)$

CORRUPT($\mathsf{C},\mathsf{S}$):
$\mathrm{CS} \leftarrow \mathrm{CS} \cup \{(\mathsf{C},\mathsf{S})\}$ ; Return $\mathsf{pw}_{\mathsf{CS}}$

REVEAL($\mathsf{P}, i$):
Require: $\tau^i_{\mathsf{P}}.\mathsf{acc} = \mathsf{true} \wedge (\mathsf{P}, i) \notin \mathrm{TS}$
$\boxed{\text{For } (\mathsf{P}', j) \text{ s.t. } \tau^j_{\mathsf{P}'}.\mathsf{tr} = \tau^i_{\mathsf{P}}.\mathsf{tr} \text{ do } \tau^j_{\mathsf{P}'}.\mathsf{fr} \leftarrow \mathsf{false}}$
Return $\tau^i_{\mathsf{P}}.\mathsf{sk}$

TEST($\mathsf{P}, i$):
Require: $\tau^i_{\mathsf{P}}.\mathsf{acc} = \mathsf{true} \wedge (\mathsf{P}, i) \notin \mathrm{TS}$
If $\mathsf{Fresh}(\tau^i_{\mathsf{P}}) = \mathsf{false}$ then return $\bot$                      // $\mathrm{G}_0$-$\mathrm{G}_1$
$\boxed{\text{If } \tau^i_{\mathsf{P}}.\mathsf{fr} = \mathsf{false} \text{ then return } \bot}$
$\mathsf{SK}_0 \leftarrow \mathrm{REVEAL}(\mathsf{P}, i)$ ; $\mathsf{SK}_1 \leftarrow_{\$} \mathcal{K}$
$\mathrm{TS} \leftarrow \mathrm{TS} \cup \{(\mathsf{P}, i)\}$ ; Return $\mathsf{SK}_d$

HASH($z$):
If $\mathrm{T_e}[z] \neq \bot$ then $\mathsf{bad} \leftarrow \mathsf{true}$ ; Return $\bot$                // $\mathrm{G}_3$
If $\mathrm{HT}[z] \neq \bot$ then return $\mathrm{HT}[z]$
$\mathrm{HT}[z] \leftarrow_{\$} \mathcal{K}$
Return $\mathrm{HT}[z]$

FIN($d'$):
Return $(d' = d)$

SENDINIT($\mathsf{C}, i, \mathsf{S}$):
Require: $\tau^i_{\mathsf{C}} = \bot$
$x \leftarrow_{\$} \mathbb{Z}_p$ ; $X^* \leftarrow g^x \cdot g^{m\cdot \mathsf{pw}_{\mathsf{CS}}}$
$\tau^i_{\mathsf{C}} \leftarrow (x, (\mathsf{C},\mathsf{S},X^*,\bot), \bot, \mathsf{false}, \boxed{\mathsf{false}})$
Return $X^*$

SENDRESP($\mathsf{S}, i, \mathsf{C}, X^*$):
Require: $\tau^i_{\mathsf{S}} = \bot$
$y \leftarrow_{\$} \mathbb{Z}_p$ ; $Y^* \leftarrow g^y \cdot g^{n\cdot \mathsf{pw}_{\mathsf{CS}}}$
If $\exists\, \mathsf{P} \in \mathcal{C} \cup \mathcal{S}$ and $j$ s.t. $\tau^j_{\mathsf{P}}.\mathsf{tr} = (\mathsf{C},\mathsf{S},X^*,Y^*)$ then      // $\mathrm{G}_1$-$\mathrm{G}_3$
    Return $\bot$                                                       // $\mathrm{G}_1$-$\mathrm{G}_3$
$\boxed{\mathsf{fr} \leftarrow [\![(\mathsf{C},\mathsf{S}) \notin \mathrm{CS}]\!]}$
$X \leftarrow X^* / g^{m\cdot \mathsf{pw}_{\mathsf{CS}}}$
$Z \leftarrow X^y$
$\mathsf{SK} \leftarrow \mathrm{HASH}(\mathsf{C},\mathsf{S},X^*,Y^*,\mathsf{pw}_{\mathsf{CS}},Z)$
$\tau^i_{\mathsf{S}} \leftarrow (y, (\mathsf{C},\mathsf{S},X^*,Y^*), \mathsf{SK}, \mathsf{true}, \boxed{\mathsf{fr}})$
Return $Y^*$

SENDTERMINIT($\mathsf{C}, i, \mathsf{S}, Y^*$):
If $\tau^i_{\mathsf{C}} \neq (x, (\mathsf{C},\mathsf{S},X^*,\bot), \bot, \mathsf{false}, \boxed{\mathsf{false}})$ for some $x, X^*$ then
    Return $\bot$
If $\exists\, \mathsf{P} \in \mathcal{C} \cup \mathcal{S} \setminus \{\mathsf{S}\}$ and $j$ s.t. $\tau^j_{\mathsf{P}}.\mathsf{tr} = (\mathsf{C},\mathsf{S},X^*,Y^*)$ then   // $\mathrm{G}_1$-$\mathrm{G}_3$
    Return $\bot$                                                       // $\mathrm{G}_1$-$\mathrm{G}_3$
$\boxed{\mathsf{fr} \leftarrow [\![\exists!\, j \text{ s.t. } (\tau^j_{\mathsf{S}}.\mathsf{tr} = (\mathsf{C},\mathsf{S},X^*,Y^*) \wedge \tau^j_{\mathsf{S}}.\mathsf{fr} = \mathsf{true})]\!]}$
$\boxed{\mathsf{fr} \leftarrow \mathsf{fr} \vee [\![(\mathsf{C},\mathsf{S}) \notin \mathrm{CS}]\!]}$
$Y \leftarrow Y^* / g^{n\cdot \mathsf{pw}_{\mathsf{CS}}}$
$Z \leftarrow Y^x$
$\mathsf{SK} \leftarrow \mathrm{HASH}(\mathsf{C},\mathsf{S},X^*,Y^*,\mathsf{pw}_{\mathsf{CS}},Z)$
$\tau^i_{\mathsf{C}} \leftarrow (x, (\mathsf{C},\mathsf{S},X^*,Y^*), \mathsf{SK}, \mathsf{true}, \boxed{\mathsf{fr}})$
Return $\bot$
```

Figure 22: Games $\mathrm{G}_0$-$\mathrm{G}_3$ for the proof of Theorem 5.3.

```
Games $\boxed{G_4,\ G_5}$

INIT:
  $d \leftarrow_{\$} \{0,1\}$ ; $m, n \leftarrow_{\$} \mathbb{Z}_p$ ; $M \leftarrow g^m$ ; $N \leftarrow g^n$
  $\pi \leftarrow (M, N)$ ; $ad \leftarrow (m, n)$
  For $(\mathsf{C}, \mathsf{S}) \in \mathcal{C} \times \mathcal{S}$ do $\mathsf{pw}_{\mathsf{CS}} \leftarrow_{\$} \mathcal{PW}$
  Return $(\pi, ad)$

EXECUTE$(\mathsf{C}, i, \mathsf{S}, j)$:
  Require: $\tau_{\mathsf{C}}^i = \perp \wedge \tau_{\mathsf{S}}^j = \perp$
  $\boxed{\bar{x} \leftarrow_{\$} \mathbb{Z}_p \ ; \ X^* \leftarrow g^{\bar{x}} \ ; \ x \leftarrow \bar{x} - m \cdot \mathsf{pw}_{\mathsf{CS}}}$
  $\boxed{\bar{y} \leftarrow_{\$} \mathbb{Z}_p \ ; \ Y^* \leftarrow g^{\bar{y}} \ ; \ y \leftarrow \bar{y} - n \cdot \mathsf{pw}_{\mathsf{CS}}}$
  If $\exists\, \mathsf{P} \in \mathcal{C} \cup \mathcal{S}$ and $j'$ s.t. $\tau_{\mathsf{P}}^{j'}.\mathsf{tr} = (\mathsf{C}, \mathsf{S}, X^*, Y^*)$ then
    Return $\perp$
  If $\mathrm{HT}[\mathsf{C}, \mathsf{S}, X^*, Y^*, \mathsf{pw}_{\mathsf{CS}}, g^{xy}] \neq \perp$ then
    Return $\perp$
  $\mathsf{SK} \leftarrow_{\$} \mathcal{K}$
  $\mathrm{T_e}[\mathsf{C}, \mathsf{S}, X^*, Y^*, \mathsf{pw}_{\mathsf{CS}}, g^{xy}] \leftarrow \mathsf{SK}$
  $\tau_{\mathsf{C}}^i \leftarrow (x, (\mathsf{C}, \mathsf{S}, X^*, Y^*), \mathsf{SK}, \mathsf{true}, \mathsf{true})$
  $\tau_{\mathsf{S}}^j \leftarrow (y, (\mathsf{C}, \mathsf{S}, X^*, Y^*), \mathsf{SK}, \mathsf{true}, \mathsf{true})$
  Return $(X^*, Y^*)$

CORRUPT$(\mathsf{C}, \mathsf{S})$:
  $\mathrm{CS} \leftarrow \mathrm{CS} \cup \{(\mathsf{C}, \mathsf{S})\}$ ; Return $\mathsf{pw}_{\mathsf{CS}}$

REVEAL$(\mathsf{P}, i)$:
  Require: $\tau_{\mathsf{P}}^i.\mathsf{acc} = \mathsf{true} \wedge (\mathsf{P}, i) \notin \mathrm{TS}$
  For $(\mathsf{P}', j)$ s.t. $\tau_{\mathsf{P}'}^j.\mathsf{tr} = \tau_{\mathsf{P}}^i.\mathsf{tr}$ do $\tau_{\mathsf{P}'}^j.\mathsf{fr} \leftarrow \mathsf{false}$
  Return $\tau_{\mathsf{P}}^i.\mathsf{sk}$

TEST$(\mathsf{P}, i)$:
  Require: $\tau_{\mathsf{P}}^i.\mathsf{acc} = \mathsf{true} \wedge (\mathsf{P}, i) \notin \mathrm{TS}$
  If $\tau_{\mathsf{P}}^i.\mathsf{fr} = \mathsf{false}$ then return $\perp$
  $\mathsf{SK}_0 \leftarrow \mathrm{REVEAL}(\mathsf{P}, i)$ ; $\mathsf{SK}_1 \leftarrow_{\$} \mathcal{K}$
  $\mathrm{TS} \leftarrow \mathrm{TS} \cup \{(\mathsf{P}, i)\}$ ; Return $\mathsf{SK}_d$

HASH$(z)$:
  If $\mathrm{T_e}[z] \neq \perp$ then return $\perp$
  $(\mathsf{C}, \mathsf{S}, X^*, Y^*, \mathsf{pw}, Z) \leftarrow z$
  If $\mathrm{T_s}[\mathsf{C}, \mathsf{S}, X^*, Y^*] = (\text{"server"}, \bar{y}, \mathsf{SK})$ then                    // $G_5$
    If $(\mathsf{pw} = \mathsf{pw}_{\mathsf{CS}}) \wedge (Z = (X^*/g^{m \cdot \mathsf{pw}})^{\bar{y} - n \cdot \mathsf{pw}})$ then   // $G_5$
      $\mathsf{bad} \leftarrow \mathsf{true}$ ; Return $\perp$                                  // $G_5$
  If $\mathrm{T_s}[\mathsf{C}, \mathsf{S}, X^*, Y^*] = (\text{"client"}, \bar{x}, \mathsf{SK})$ then                    // $G_5$
    If $(\mathsf{pw} = \mathsf{pw}_{\mathsf{CS}}) \wedge (Z = (Y^*/g^{n \cdot \mathsf{pw}})^{\bar{x} - m \cdot \mathsf{pw}})$ then   // $G_5$
      $\mathsf{bad} \leftarrow \mathsf{true}$ ; Return $\perp$                                  // $G_5$
  If $\mathrm{HT}[z] \neq \perp$ then return $\mathrm{HT}[z]$
  $\mathrm{HT}[z] \leftarrow_{\$} \mathcal{K}$
  Return $\mathrm{HT}[z]$

FIN$(d')$:
  Return $(d' = d)$
```

```
SENDINIT$(\mathsf{C}, i, \mathsf{S})$:
  Require: $\tau_{\mathsf{C}}^i = \perp$
  $\boxed{\bar{x} \leftarrow_{\$} \mathbb{Z}_p \ ; \ X^* \leftarrow g^{\bar{x}} \ ; \ x \leftarrow \bar{x} - m \cdot \mathsf{pw}_{\mathsf{CS}}}$
  $\tau_{\mathsf{C}}^i \leftarrow (x, (\mathsf{C}, \mathsf{S}, X^*, \perp), \perp, \mathsf{false}, \mathsf{false})$
  Return $X^*$

SENDRESP$(\mathsf{S}, i, \mathsf{C}, X^*)$:
  Require: $\tau_{\mathsf{S}}^i = \perp$
  $\boxed{\bar{y} \leftarrow_{\$} \mathbb{Z}_p \ ; \ Y^* \leftarrow g^{\bar{y}} \ ; \ y \leftarrow \bar{y} - n \cdot \mathsf{pw}_{\mathsf{CS}}}$
  If $\exists\, \mathsf{P} \in \mathcal{C} \cup \mathcal{S}$ and $j$ s.t. $\tau_{\mathsf{P}}^j.\mathsf{tr} = (\mathsf{C}, \mathsf{S}, X^*, Y^*)$ then
    Return $\perp$
  $\mathsf{fr} \leftarrow [\![(\mathsf{C}, \mathsf{S}) \notin \mathrm{CS}]\!]$
  $X \leftarrow X^*/g^{m \cdot \mathsf{pw}_{\mathsf{CS}}}$
  $\boxed{Z \leftarrow X^{\bar{y} - n \cdot \mathsf{pw}_{\mathsf{CS}}}}$
  If $\neg\, \mathsf{fr}$ then                                                       // $G_5$
    $\mathsf{SK} \leftarrow \mathrm{HASH}(\mathsf{C}, \mathsf{S}, X^*, Y^*, \mathsf{pw}_{\mathsf{CS}}, Z)$                  // $G_5$
  Else                                                                  // $G_5$
    If $\mathrm{HT}[\mathsf{C}, \mathsf{S}, X^*, Y^*, \mathsf{pw}_{\mathsf{CS}}, Z] \neq \perp$ then                  // $G_5$
      $\mathsf{bad} \leftarrow \mathsf{true}$ ; Return $\perp$                                  // $G_5$
    $\mathsf{SK} \leftarrow_{\$} \mathcal{K}$                                                  // $G_5$
    $\mathrm{T_s}[\mathsf{C}, \mathsf{S}, X^*, Y^*] \leftarrow (\text{"server"}, \bar{y}, \mathsf{SK})$                    // $G_5$
  $\mathsf{SK} \leftarrow \mathrm{HASH}(\mathsf{C}, \mathsf{S}, X^*, Y^*, \mathsf{pw}_{\mathsf{CS}}, Z)$                    // $G_4$
  $\tau_{\mathsf{S}}^i \leftarrow (y, (\mathsf{C}, \mathsf{S}, X^*, Y^*), \mathsf{SK}, \mathsf{true}, \mathsf{fr})$
  Return $Y^*$

SENDTERMINIT$(\mathsf{C}, i, \mathsf{S}, Y^*)$:
  If $\tau_{\mathsf{C}}^i \neq (x, (\mathsf{C}, \mathsf{S}, X^*, \perp), \perp, \mathsf{false}, \mathsf{false})$ for some $x, X^*$ then
    Return $\perp$
  If $\exists\, \mathsf{P} \in \mathcal{C} \cup \mathcal{S} \setminus \{\mathsf{S}\}$ and $j$ s.t. $\tau_{\mathsf{P}}^j.\mathsf{tr} = (\mathsf{C}, \mathsf{S}, X^*, Y^*)$ then
    Return $\perp$
  $\mathsf{fr} \leftarrow [\![\exists!\, j \text{ s.t. } (\tau_{\mathsf{S}}^j.\mathsf{tr} = (\mathsf{C}, \mathsf{S}, X^*, Y^*) \wedge \tau_{\mathsf{S}}^j.\mathsf{fr} = \mathsf{true})]\!]$
  $\mathsf{fr} \leftarrow \mathsf{fr} \vee [\![(\mathsf{C}, \mathsf{S}) \notin \mathrm{CS}]\!]$
  $Y \leftarrow Y^*/g^{n \cdot \mathsf{pw}_{\mathsf{CS}}}$
  $\boxed{\bar{x} \leftarrow x + m \cdot \mathsf{pw}_{\mathsf{CS}} \ ; \ Z \leftarrow Y^{\bar{x} - m \cdot \mathsf{pw}_{\mathsf{CS}}}}$
  If $\neg\, \mathsf{fr}$ then                                                       // $G_5$
    $\mathsf{SK} \leftarrow \mathrm{HASH}(\mathsf{C}, \mathsf{S}, X^*, Y^*, \mathsf{pw}_{\mathsf{CS}}, Z)$                  // $G_5$
  Else if $\mathrm{T_s}[\mathsf{C}, \mathsf{S}, X^*, Y^*] \neq \perp$                                  // $G_5$
    $(\cdot, \cdot, \mathsf{SK}) \leftarrow \mathrm{T_s}[\mathsf{C}, \mathsf{S}, X^*, Y^*]$                              // $G_5$
  Else                                                                  // $G_5$
    If $\mathrm{HT}[\mathsf{C}, \mathsf{S}, X^*, Y^*, \mathsf{pw}_{\mathsf{CS}}, Z] \neq \perp$ then                  // $G_5$
      $\mathsf{bad} \leftarrow \mathsf{true}$ ; Return $\perp$                                  // $G_5$
    $\mathsf{SK} \leftarrow_{\$} \mathcal{K}$                                                  // $G_5$
    $\mathrm{T_s}[\mathsf{C}, \mathsf{S}, X^*, Y^*] \leftarrow (\text{"client"}, \bar{x}, \mathsf{SK})$                    // $G_5$
  $\mathsf{SK} \leftarrow \mathrm{HASH}(\mathsf{C}, \mathsf{S}, X^*, Y^*, \mathsf{pw}_{\mathsf{CS}}, Z)$                    // $G_4$
  $\tau_{\mathsf{C}}^i \leftarrow (x, (\mathsf{C}, \mathsf{S}, X^*, Y^*), \mathsf{SK}, \mathsf{true}, \mathsf{fr})$
  Return $\perp$
```

Figure 23: Games $G_4$-$G_5$ for the proof of Theorem 5.3. The boxed code is included in both $G_4$ and $G_5$, and indicates a difference from $G_3$ of the prior figure.

```
Adversary $B(X_1, \ldots, X_{q_e}, Y_1, \ldots, Y_{q_e})$

InitSim:
$d \leftarrow\$ \{0,1\}$ ; $m, n \leftarrow\$ \mathbb{Z}_p$ ; $M \leftarrow g^m$ ; $N \leftarrow g^n$ ; $k \leftarrow 0$
$\pi \leftarrow (M, N)$ ; $ad \leftarrow (m, n)$
For $(\mathsf{C}, \mathsf{S}) \in \mathcal{C} \times \mathcal{S}$ do $\mathsf{pw}_{\mathsf{CS}} \leftarrow\$ \mathcal{PW}$
Return $(\pi, ad)$

ExecuteSim$(\mathsf{C}, i, \mathsf{S}, j)$:
Require: $\tau_{\mathsf{C}}^i = \bot \wedge \tau_{\mathsf{S}}^j = \bot$
$k \leftarrow k + 1$
$X^* \leftarrow X_k \cdot g^{m \cdot \mathsf{pw}_{\mathsf{CS}}}$
$Y^* \leftarrow Y_k \cdot g^{n \cdot \mathsf{pw}_{\mathsf{CS}}}$
If $\exists\, \mathsf{P} \in \mathcal{C} \cup \mathcal{S}$ and $j'$ s.t. $\tau_{\mathsf{P}}^{j'}.\mathsf{tr} = (\mathsf{C}, \mathsf{S}, X^*, Y^*)$ then
    Return $\bot$
If $\mathrm{T_{cdh}}[\mathsf{C}, \mathsf{S}, X^*, Y^*, \mathsf{pw}_{\mathsf{CS}}] \neq \bot$ then
    $Z \leftarrow \mathrm{T_{cdh}}[\mathsf{C}, \mathsf{S}, X^*, Y^*, \mathsf{pw}_{\mathsf{CS}}]$
    $\mathbf{G}^{\mathrm{scdh}}_{\mathbb{G},p,g,q_e}.\textsc{Fin}(k, Z)$
$\mathsf{SK} \leftarrow\$ \mathcal{K}$
$\mathrm{T_e}[\mathsf{C}, \mathsf{S}, X^*, Y^*, \mathsf{pw}_{\mathsf{CS}}, k] \leftarrow \mathsf{SK}$
$\tau_{\mathsf{C}}^i \leftarrow (x, (\mathsf{C}, \mathsf{S}, X^*, Y^*), \mathsf{SK}, \mathsf{true}, \mathsf{true})$
$\tau_{\mathsf{S}}^j \leftarrow (y, (\mathsf{C}, \mathsf{S}, X^*, Y^*), \mathsf{SK}, \mathsf{true}, \mathsf{true})$
Return $(X^*, Y^*)$

HSim$(z)$:
$(\mathsf{C}, \mathsf{S}, X^*, Y^*, \mathsf{pw}, Z) \leftarrow z$
If $\mathsf{pw} = \mathsf{pw}_{\mathsf{CS}}$ then
    Set $k'$ to be the smallest index in $[1..q_e]$ s.t. $(X^*/M^{\mathsf{pw}}, Y^*/N^{\mathsf{pw}}) = (X_{k'}, Y_{k'})$
    If $k' \neq \bot$ and if $\mathrm{DDH}(k', Y_{k'}, Z) = \mathsf{true}$ then
        If $k' \leq k$ then $\mathbf{G}^{\mathrm{scdh}}_{\mathbb{G},p,g,q_e}.\textsc{Fin}(k', Z)$
        If $k' > k$ then $\mathrm{T_{cdh}}[\mathsf{C}, \mathsf{S}, X^*, Y^*, \mathsf{pw}] \leftarrow Z$
If $\mathrm{HT}[z] \neq \bot$ then return $\mathrm{HT}[z]$
$\mathrm{HT}[z] \leftarrow\$ \mathcal{K}$
Return $\mathrm{HT}[z]$
```

Figure 24: Adversary $B$ for proof of Theorem 5.3. $B$ runs $A[\textsc{InitSim}, \textsc{ExecuteSim}, \textsc{HSim}, \ldots]$ with access to subroutines InitSim, ExecuteSim and HSim as shown. The remaining Corrupt, Reveal, Test and Send oracles are implemented as subroutines matching game $\mathrm{G}_3$ of Figure 22.