A preliminary version of this paper appears in the proceedings of Asiacrypt 2024. This is the full version.

The Concrete Security of Two-Party Computation: Simple Definitions, and Tight Proofs for PSI and OPRFs

Mihir Bellare[1] Rishabh Ranjan[2] Doreen Riepel[3] Ali Aldakheel[4]

July 2025

Abstract

This paper initiates a concrete-security treatment of two-party secure computation. The first step is to propose, as target, a simple, indistinguishability-based definition that we call InI. This could be considered a poor choice if it were weaker than standard simulation-based definitions, but it is not; we show that for functionalities satisfying a condition called invertibility, that we define and show is met by functionalities of practical interest like PSI and its variants, the two definitions are equivalent. Based on this, we move forward to study the concrete security of a canonical OPRF-based construction of PSI, giving a tight proof of InI security of the constructed PSI protocol based on the security of the OPRF. This leads us to the concrete security of OPRFs, where we show how different DH-style assumptions on the underlying group yield proofs of different degrees of tightness, including some that are tight, for the well-known and efficient 2H-DH OPRF, and thus for the corresponding DH PSI protocol. We then give a new PSI protocol, called salted-DH PSI, that is as efficient as DH-PSI, yet enjoys tighter proofs.

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

[2] Department of Computer Science & Engineering, University of California San Diego, USA. Email: riranjan@ucsd.edu. URL: ranjan-rishabh.github.io/. Supported in part by NSF grant CNS-2154272 and KACST.

[3] Department of Computer Science & Engineering, University of California San Diego, USA. Email: doreen.riepel@gmail.com. URL: doreenriepel.me/. Supported in part by KACST.

[4] Center of Excellence for Secure Computing, King Abdulaziz City for Science and Technology (KACST), Saudi Arabia. Email: amaldakheel@kacst.edu.sa.

Contents

1 Introduction

The first wave of research on secure two-party computation (2PC) [56] asked whether this magical-sounding goal could even be achieved. The focus being feasibility rather than efficiency, results were given in an asymptotic security framework and reduction tightness was not a concern. We are now in a second wave, fueled by real-world applications, where the focus is efficient protocols for particular goals like PSI and OPRFs. We suggest that, in this second wave, we need results in a concrete security framework, and reductions as tight as possible, so that we can find and pick the protocols that give the best efficiency for a desired level of proven security.

CONTRIBUTIONS IN BRIEF. Towards the above, this paper initiates a concrete-security treatment of 2PC. It has two main parts:

1. Definitional framework. We give and target a new, simple and concrete-security friendly definition of security for 2PC that we call input indistinguishability (InI). As the name indicates, it is indistinguishability based. Yet, for functionalities satisfying a condition called invertibility, that we define and is met by functionalities of practical interest including PSI and friends, we show that InI is as strong as standard simulation-based definitions. Our definitional framework explicitly incorporates random oracles and surfaces some subtleties in this regard.

2. Results for PSI and OPRFs. We consider the concrete security of OPRF-based PSI [32], giving a tight proof of InI security of the constructed PSI protocol based on the security of the starting OPRF. This motivates studying the concrete security of OPRFs, where we show how different DH-style assumptions on the underlying group yield proofs of different degrees of tightness, including some that are tight, for the well-known and efficient 2H-DH OPRF [36], and thus for DH-PSI, the PSI protocol based on 2H-DH. We follow this with a new protocol, salted DH PSI, for which we give tighter proofs. Salted DH PSI is essentially as efficient as DH PSI, showing how concrete security can be improved through protocol changes.

1.1 Setting the stage

THE ASYMPTOTIC SETTING. Provable security [30, 16] began in an *asymptotic framework* inherited from computational complexity theory. To show that a scheme Π meets a target notion of security T assuming that an underlying problem P (e.g. CDH) is hard, one gives a reduction that takes an adversary A and builds an adversary A' such that:

> If A is PPT and has non-negligible advantage (success probability) $\epsilon_A(\cdot)$ in violating T-security of Π, then A' is also PPT and has non-negligible advantage $\epsilon_{A'}(\cdot)$ in breaking P.

The advantages here are functions of a security parameter k, and such a function is "negligible" if it goes to zero faster than the reciprocal of any polynomial. Such results help build theoretical foundations but give implementors no explicit way to pick the security parameter to guarantee a desired level (e.g. 256 bits) of security.

THE CONCRETE SETTING. In the *concrete framework* [10], one continues to give a reduction that takes an adversary A and builds an adversary A', but now additionally specifying a function $\mathbf{B}$, called the *bound*, such that:

> If A has running time t, resources $\mathbf{R}$ and advantage $\mathbf{Adv}^{\mathrm{T}}_{\Pi}(A)$ in violating T-security of Π, then A' has running time about t and advantage ϵ' in breaking P such that $\mathbf{Adv}^{\mathrm{T}}_{\Pi}(A) \leq \mathbf{B}(\epsilon', \mathbf{R})$.

The advantages here are real numbers, not functions. There is no explicit security parameter. Resources include the number of queries to various oracles in the game defining security. A reduction is *tight* if $\mathbf{B}(\epsilon', \mathbf{R}) = c \cdot \epsilon'$ for some small constant c. A typical example of a non-tight reduction is

$\mathbf{B}(\epsilon', \mathbf{R}) = q \cdot \epsilon'$ where $q \in \mathbf{R}$ is the number of queries of A to some oracle. Now if an implementor wants to ensure $\mathbf{Adv}^{\mathrm{T}}_{\Pi}(A) \leq \epsilon$ for some choices of $t, \epsilon, \mathbf{R}$, they can use the bound to determine ϵ' such that $\epsilon \leq \mathbf{B}(\epsilon', \mathbf{R})$ and then use t, ϵ' to make a choice of group or elliptic curve in which to work. The tighter the reduction, the smaller is ϵ' and thus the size of the curve, and the more efficient is the implemented scheme.

For example, suppose Π_1, Π_2 are protocols for some goal (say PSI), both proven secure under the CDH assumption over an underlying elliptic curve group, the first using six modular exponentiations in the group and second only three. Is Π_2 the one to prefer and implement? Not necessarily. The right comparison is at the same level of concrete security for both, say 128-bits. To provide this, say that, due to different degrees of tightness in the proofs for the two protocols, we need a 256-bit curve $\mathbb{G}_{256}$ for Π_1 and a 384-bit curve $\mathbb{G}_{384}$ for Π_2. Then (since exponentiation is cubic-time) Π_2 is $(3/6) \cdot (384/256)^3 = 1.6875$ times *more* expensive than Π_1, despite using *fewer* exponentiations in the group, making Π_1 the sounder choice.

Thus, beyond allowing sound choices of parameters, the concrete framework leads to new questions, such as to seek tighter reductions for existing protocols or to seek new protocols which allow tight reductions. These kinds of questions (which we will pursue for 2PC) are invisible in the asymptotic setting.

Concrete security is not new. In provable-security for symmetric cryptography, it is the norm, and it is widely employed in public-key cryptography and authenticated key-exchange.

Definitional complexity. Our intent is to facilitate and provide concrete security assessments and improvements for 2PC. The first step is simple, "concrete-security-friendly" definitions. We start with a broad definitional classification aimed at saying what this means.

Having fixed a target notion T and scheme or protocol Π, our discussions of security above assumed a simple definitional format in which the advantage $\mathbf{Adv}^{\mathrm{T}}_{\Pi}(A)$ is associated to just the adversary A, as is true for basic notions like UF-CMA (signatures), PRF-security or indistinguishability of encryptions. We will call these single-quantifier definitions since the security requirement is

$$\forall\, A \,:\, \mathbf{Adv}^{\mathrm{T}}_{\Pi}(A) \text{ is low.}$$

In a simulation-based definition, however, the advantage $\mathbf{Adv}^{\mathrm{T}}_{\Pi,\mathsf{S}}(A)$ is now relative to a simulator S. The requirement now being

$$\forall\, A\, \exists\, \mathsf{S} \,:\, \mathbf{Adv}^{\mathrm{T}}_{\Pi,\mathsf{S}}(A) \text{ is low} \quad \text{or} \quad \exists\, \mathsf{S}\, \forall A \,:\, \mathbf{Adv}^{\mathrm{T}}_{\Pi,\mathsf{S}}(A) \text{ is low,}$$

we refer to these as double-quantifier definitions. But this double-quantifier structure does not fit the above-discussed format and complicates concrete-security assessments.

To elaborate, double-quantifier definitions do not preclude giving concrete-security results, and there are some in the literature. (Examples include [55, Theorem 4] and [36, Theorem 1].) However it is not clear (at least to us) how to use such results to pick parameters to guarantee a desired level of security. One issue is that we would expect $\mathbf{Adv}^{\mathrm{T}}_{\Pi,\mathsf{S}}(A)$ to be lower for simulators with higher running time, raising the question of how to interpret this advantage and also making parameter choice depend on simulator running time. Also complexity grows when (as happens in the first just-cited result), one simulator is defined in terms of another, making it hard to work in a modular way.

Concrete-security friendliness is not the only benefit of single-quantifier definitions. Another is attack-friendliness. To give an attack, we need only give an adversary with high advantage. In a double-quantifier definition, we would have to prove that the advantage is high relative to *all* simulators. Our InI definition in particular facilitates cryptanalysis of 2PC protocols, a topic

largely unexplored.

2PC. Recall that the setting of secure two-party computation (2PC) considers parties 1, 2 (also called client and server, respectively) having inputs x_1, x_2 respectively. A 2-party protocol Π, to securely compute a functionality F, allows the parties to interact so that at the end they have outputs y_1, y_2 respectively, where $(y_1, y_2) \leftarrow \mathsf{F}(x_1, x_2)$. Yet, neither party should learn more about the other party's input than disclosed by the output they obtain.

This area has traditionally used double-quantifier definitions in an asymptotic framework. But today the quest is efficient protocols for goals (functionalities) of interest in applications, where (for reasons given above) a concrete framework is crucial. Leading the way, concrete-security results for single-quantifier definitions have been given for garbling schemes [8]. However there are many protocols, for goals including PSI, OPRF and their variants, which target efficiency without concrete security. We aim to fill this gap.

1.2 Our definitional framework

The highlight of our framework below will be a definition for 2PC security, called InI, that is (1) single-quantifier and thus both concrete-security friendly and attack-friendly, yet (2) usually no less powerful than a standard (double-quantifier) simulation-based definition.

SIM and InI. We want to say (in a concrete framework) what it means for a protocol Π to securely compute a 2PC functionality F. The classical paradigm for 2PC definitions is simulation [43, 19], so we start there, giving a concretely-rendered definition, called SIM. It defines an advantage $\mathbf{Adv}^{\mathrm{sim}}_{\mathsf{F},\Pi,\mathsf{S}}(A)$ for an adversary A relative to a simulator S. As above, this is a double-quantifier definition but represents an important baseline, in terms of strength and history, that we want to respect.

Alongside, we give a simple, single-quantifier definition that we call *input indistinguishability* (InI). Let $\mathsf{F}(x_1, x_2)[i]$ denote the output given by the functionality to party i. Suppose party 1 is the "honest" one, meaning the one whose privacy we aim to protect. Party 2, as the adversary A, supplies a pair of inputs $x_{1,0}, x_{1,1}$ for party 1 and a single input x_2 for itself such that the outputs $y_0 = \mathsf{F}(x_{1,0}, x_2)[2]$ and $y_1 = \mathsf{F}(x_{1,1}, x_2)[2]$ for itself are the same. A random challenge bit b is chosen, and now A, given its view (transcript and coins) of the execution of protocol Π on inputs $x_{1,b}, x_2$, outputs a guess b'. InI asks that its advantage $\mathbf{Adv}^{\mathrm{InI}}_{\mathsf{F},\Pi}(A) = 2\Pr[b = b'] - 1$ is small. A formal definition is in Section 3.

Relating SIM and InI. We propose to use InI in concrete-security results and parameter choices. As the reader may note, this would be a poor choice if InI is weaker (provides less security) than SIM, but we show that, for functionalities of practical interest, this is not the case and in fact the two are equivalent. To elaborate, Theorem 3.4 says that InI implies SIM (that is, any Π that is InI-secure is also SIM-secure) as long as the target functionality F satisfies a condition we define called *invertibility*. The latter (continuing to assume party 1 is the honest one) asks that, given x_2, y_2, it is possible to efficiently find an input x_1 for party 1 satisfying $\mathsf{F}(x_1, x_2)[2] = y_2$, assuming of course such an x_1 exists. In the other direction, Theorem 3.2 says that SIM always implies InI.

In Section 3.3, we show that the functionality for PSI is invertible, as are variants of it like for threshold-PSI and cardinality-PSI [26]. We also show, in Appendix E, invertibility for Oblivious Transfer (OT) [52] and the Secure Inferencing functionality of [39]. So for all these we may safely focus on InI, reassured that it is qualitatively just as strong as SIM.

We clarify and caution that it is not the case that InI and SIM are equivalent for *all* functionalities. Indeed, in Theorem 3.3 we give a counterexample, meaning a (non-invertible) F and a protocol Π such that Π is InI-secure, but not SIM-secure, for F. However, the F, Π here are contrived and

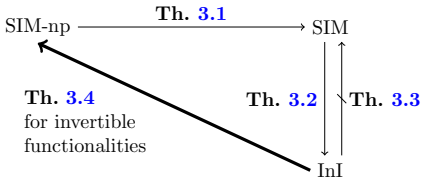


Figure 1: Relations between the InI, SIM and SIM-np notions of security for a 2PC protocol Π for a functionality F. Arrows are implications and the barred arrow is a separation.

artificial. Our experience is that natural functionalities of practical interest tend to be invertible and thus enjoy the equivalence of InI and SIM guaranteed by Theorems 3.2, 3.4.

ROM incorporation and subtleties. The Random Oracle (RO) Model [13] is extensively employed for practical 2PC but, while used in proofs, the RO is sometimes absent in the definitions. Our definitions in contrast explicitly and flexibly incorporate random oracles. Protocols name a space from which their desired RO is then drawn in games defining security. In the default SIM notion, the RO is programmable by the simulator. We also give a non-programmable RO version SIM-np. InI has the attractive, and further simplifying feature that the RO is inherently non-programmable. (There is no simulator to program it.)

Attending carefully to formalizing the ROM usage in these definitions brought to light a subtle issue. RO queries could be made not only by the protocol and adversary, but also by the functionality. We show, by example, that allowing the simulator to program the answers to functionality RO queries is problematic and can lead to clearly insecure protocols having a proof of security. Our SIM definition addresses this, answering functionality queries via an honestly chosen random function that is then given as oracle to the simulator, who can use it, or not, as it likes, in answering other RO queries. (See Section 3.1 for details.)

Full relations picture. With that, Figure 1 summarizes the full set of relations between the notions. An arrow $\mathrm{X} \to \mathrm{Y}$ is an implication, meaning any Π that is X-secure for F is also Y-secure for F. A barred arrow $\mathrm{X} \not\to \mathrm{Y}$ is a separation, meaning there exist F, Π such that Π is X-secure for F but not Y-secure for F. For invertible F, we note that Theorem 3.4 actually shows InI $\to$ SIM-np, which implies InI $\to$ SIM because Theorem 3.1 says that SIM-np $\to$ SIM.

The setting. Our definitions and results are in the semi-honest (also called honest-but-curious) setting, where the parties aim to learn each other's inputs but are assumed to follow the protocol. While we want to eventually treat the malicious case, there are several reasons to start with the semi-honest one. The first is pedagogic; as our work shows, the semi-honest case is hardly trivial, and jumping to the malicious case without a solid foundation for the semi-honest one felt to us premature and unsound. The second reason is that many works in the literature [25, 51, 40, 21, 41, 50, 20] give protocols for the semi-honest setting, and understanding their concrete security is important for practical reasons. Pragmatic concerns too justify this setting. The gain is efficiency; malicious-secure protocols are typically more expensive, which may curtail adoption. Meanwhile, with regard to security, in practice there are forces external to the cryptography that deter malicious behavior. Parties are often corporations who are subject to laws and bound by contracts with other parties. Use of subverted (malicious) code risks discovery and exposure. Add to this that the protocol functionality is already giving these parties the information they want, and malicious behavior emerges as both low reward and high risk.

1.3 Concrete-Security results for 2PC protocols

We give some general results, and then focus on PSI and OPRFs.

MANY EXECUTIONS VERSUS ONE. In practice we expect that a protocol Π is executed many times, on different inputs. Our definitions accordingly allow the adversary to obtain as many execution transcripts as it likes via queries to an oracle RUN. In the concrete setting we are interested in how adversary advantage degrades as a function of the number $q_{\rm rn}$ of queries to RUN. Theorem 3.5 confirms that the hybrid argument works as expected to show that the advantage $\epsilon^{\rm InI}_{\Pi}(q_{\rm rn})$ for $q_{\rm rn}$ queries is at most $q_{\rm rn}$ times the advantage $\epsilon^{\rm InI}_{\Pi}(1)$ for one query.

In an asymptotic setting, the question would end here, but concretely, it is more of a starting point, raising the question of whether we can, for particular protocols, avoid the $q_{\rm rn}$ factor loss, meaning show that $\epsilon^{\rm InI}_{\Pi}(q_{\rm rn}) \approx \epsilon^{\rm InI}_{\Pi}(1)$. The following will show (amongst other things) that the answer is yes.

PSI FROM OPRFS, TIGHTLY. In Private Set Intersection (PSI) [26], the inputs x_1, x_2 are sets and the functionality $\mathsf{F}^{\rm psi}$ returns their intersection $x_1 \cap x_2$ to the client and nothing to the server. PSI is used for privacy-respecting solutions in the following domains: ad conversion [35, 34], contact discovery [44], password or credential monitoring [42, 54, 4, 33], genomics [5], proximity testing [47], relationship discovery in social networks [45] and detection of sexual misconduct [53]. These applications motivate PSI protocols with tight proofs.

Towards this, we focus on one simple, canonical way to achieve PSI suggested by Hazay and Lindell [32], where the PSI protocol $\Pi^{\rm psi}$ is built from a protocol $\Pi^{\rm oprf}$ for an Oblivious Pseudo-Random Function (OPRF) [46, 25, 38]. Recall that in the latter, the server has a (secret) key K for a (regular) PRF Q, the client has an input x and the protocol ends with them holding ε and $\mathsf{Q}(K, x)$, respectively. Simulation-based (hence double-quantifier) definitions of security for OPRFs are given in [55]. We give instead single-quantifier (non-simulation-based) definitions. For client security (honest party 1), it is simply InI. For server security (honest party 2) we give a simple pseudo-randomness definition that we call OPRF-PR. Under these assumptions, Theorem 4.2 shows client and server InI security of $\Pi^{\rm psi}$. The reductions are all tight. This is true regardless of the number $q_{\rm rn}$ of $\Pi^{\rm psi}$-executions (formally, RUN queries of the adversary), meaning the bounds do not have the multiplicative $q_{\rm rn}$ factor loss of the hybrid argument of Theorem 3.5. Theorem 4.2 also separately shows correctness of $\Pi^{\rm psi}$, based on the PRF-security of Q. (The actual result is more general.)

Having thus stepped *tightly* from OPRFs to PSI, we turn to studying the concrete security of the former.

BOUNDS FOR 2H-DH OPRF AND DH-PSI. OPRFs have applications beyond PSI [22, 36, 37, 23], making their concrete security of interest in its own right. 2H-DH (Two-Hash Diffie-Hellman) [36] is the de-facto standard OPRF and thus the natural candidate to study.

Jarecki, Kiayias and Krawczyk [36, Theorem 1] prove that 2H-DH achieves a simulation-based (UC) definition in the ROM, assuming hardness of the One-More Gap Diffie-Hellman (OM-Gap-DH) problem. This is a strong assumption, giving the adversary a CDH oracle in the One-More style [11, 17] as well as a DDH oracle in the Gap style [48]. Their result is semi-concrete (a bound is given but the runtimes of the simulator and constructed adversaries are not), and the bound is not tight.

We revisit the 2H-DH OPRF and evaluate security under our (single quantifier) definitions, namely InI for client (party 1) and OPRF-PR for server (party 2), as needed for our application to PSI above. Theorem 6.1 shows client InI-security unconditionally and with a good bound. Our discussion focuses on OPRF-PR. We consider a variety of choices for the starting (assumed hard) problem P in the group $\mathbb{G}$. What we consider interesting is that we can prove security under *all* these assumptions, but with *different tightness*.

The results are given in Theorem 6.2 and summarized in the "2H-DH OPRF" column of Figure 2.

Problem P	$\mathbf{B}(\epsilon', \{q_{\mathrm{ro}}, q_{\mathrm{n}}\})$	$\mathbf{B}(\epsilon', \{q_{\mathrm{ro}}, q_{\mathrm{rn}}\})$	
	2H-DH OPRF	DH-PSI	Salted DH-PSI
CDH	$q_{\mathrm{ro}}^2 q_{\mathrm{n}} \cdot \epsilon'$	$q_{\mathrm{ro}}^2 q_{\mathrm{rn}} \cdot \epsilon'$	$q_{\mathrm{ro}} \cdot \epsilon'$
V-CDH	$q_{\mathrm{ro}} q_{\mathrm{n}} \cdot \epsilon'$	$q_{\mathrm{ro}} q_{\mathrm{rn}} \cdot \epsilon'$	ϵ'
CDH-MUC	$q_{\mathrm{ro}} \cdot \epsilon'$	$q_{\mathrm{ro}} \cdot \epsilon'$	$q_{\mathrm{ro}} \cdot \epsilon'$
V-CDH-MUC	ϵ'	ϵ'	ϵ'
DDH	ϵ'	ϵ'	ϵ'

Figure 2: **Our results for the 2H-DH OPRF and the DH-PSI and Salted DH-PSI protocols.** For different choices of the assumed-hard problem P, the 2nd column shows the bound $\mathbf{B}(\epsilon', \{q_{\mathrm{ro}}, q_{\mathrm{n}}\})$ on the oprf-pr advantage of an adversary A for the 2H-DH OPRF, while the 3rd and 4th columns show the bound $\mathbf{B}(\epsilon', \{q_{\mathrm{ro}}, q_{\mathrm{rn}}\})$ on the ini-advantage of an adversary A for the DH-PSI and Salted DH-PSI protocols, respectively, in all cases as a function of the advantage $\epsilon' = \mathbf{Adv}^{\mathrm{P}}_{\mathbb{G}}(A')$ of the constructed adversary A' in solving problem P in group $\mathbb{G}$. In the first case, $q_{\mathrm{ro}}, q_{\mathrm{n}}$ are the number of queries A makes to its random and NEW oracles, respectively, and in the other cases, $q_{\mathrm{ro}}, q_{\mathrm{rn}}$ are the number of queries A makes to its random and RUN oracles, respectively.

It considers an OPRF-PR adversary A making q_{ro} queries to its random oracle and performing q_{n} executions (formally, queries to an oracle called NEW) of the OPRF protocol. Column "2H-DH OPRF" of the table then shows (approximate) bounds on the oprf-pr advantage of A as a function of $q_{\mathrm{ro}}, q_{\mathrm{n}}$ and the advantage ϵ' of a constructed adversary A' in solving problem P in group $\mathbb{G}$.

Row 1 of the table says that we can prove security already assuming hardness of only the (plain) CDH problem. But we incur a substantial factor loss in the bound. Now we consider strengthening the assumption. First, we give the adversary a limited DDH oracle. The resulting assumption, which we call V-CDH for verifiable CDH, is weaker than either Strong-CDH [1] or Gap-CDH [48]. Row 2 shows that the factor loss in the bound drops. Second, for both CDH and V-CDH, we move from the single-user setting to the one of multiple users with corruptions. Rows 2, 3 show further drops in the bound. Finally (row 5) we give a *tight* reduction from DDH. We refer to Section 2 for formal definitions of the computational problems and the relations between them, and to Figure 13 for a more precise and complete summary of the results.

Now, let DH-PSI denote the above-discussed PSI protocol when the OPRF is set to the 2H-DH one. Then, combining the above with Theorem 4.2 gives bounds on the server InI security of DH-PSI as shown in the "DH-PSI" column of Figure 2.

SALTED DH-PSI. Concrete security raises new questions invisible in the asymptotic setting, in this case whether there is a different protocol, ideally as efficient as DH-PSI, yet with bounds better than shown for the latter in Figure 2. We show that the answer is yes, giving in Section 7 what we call the salted DH-PSI protocol. The bounds, as per Theorem 7.1 and summarized in the "Salted DH-PSI" column of Figure 2, are improved under the CDH and V-CDH assumptions and maintained under the others. The salting technique we use originates in PSS [14], a modification of the RSA-FDH signature scheme which improved the bound, for UF-CMA under the RSA assumption, from loose to tight. We warn that the bounds in the table are approximate; more precise ones can be found in Figure 15.

1.4 Discussion and further related work

An indistinguishability-style definition for garbling schemes was given in [9], and one for multi-party computation in [3]. Our InI definition was inspired by, and generalizes, an indistinguishability-based definition for threshold-PSI from [6]. InI and SIM for 2PC can be seen as analogues of witness-indistinguishability [24] and zero-knowledge [31], respectively, for proof systems. Another domain

in which both indistinguishability-style and simulation-style definitions have been given, related and used is functional encryption (FE) [18, 49, 12].

What we call single-quantifier and double-quantifier definitions are sometimes referred to as game-based and simulation-based, respectively. However games are a descriptive language and our SIM definition is also written as a game, so to avoid confusion we are using a different terminology that we feel highlights the essential difference, namely the quantifier structure.

There is a divide, in the cryptographic community, between those who speak and use the language of UC [19], and those who don't. A consequence has been to exclude a certain, and more applied part of our community, from 2PC research. Part of the intent of our work is to bridge this gap. With InI and concrete security, we have cast 2PC in a language and style similar to that used in practice-oriented work on conventional primitives like encryption, signatures and authenticated key exchange, primitives that have in particular seen a large quantity of work on proof tightness. The hope is to draw this segment of the community into 2PC to likewise explore and improve proof tightness.

In writing our definitions, we have aimed for precision, and attention to detail, at a level that to us is beyond the norm for the area. This is in part a response to our experience (admittedly perhaps due to our lack of expertise) of struggling to understand, and finding ambiguous, some definitions we try to read in the literature. A price paid is notation. Our work could (rightly) be critiqued as notationally heavy, but we believe the notation is central to greater precision and reduced ambiguity, and hope that, after some exposure, it ceases to be a significant barrier for a reader.

2 Preliminaries

NOTATION. If $\boldsymbol{w}$ is a vector then $|\boldsymbol{w}|$ is its length (the number of coordinates) and $\boldsymbol{w}[i]$ is its i-th coordinate. The empty (length zero) vector is denoted ε. We say that $\boldsymbol{w}$ is an n-vector if $|\boldsymbol{w}| = n$. We let $\mathsf{V2S}(\boldsymbol{w}) = \{\boldsymbol{w}[1], \ldots, \boldsymbol{w}[|\boldsymbol{w}|]\}$ be the set of elements of vector $\boldsymbol{w}$. Likewise, if S is a set, then $\boldsymbol{w} \leftarrow \mathsf{S2V}(S)$ puts its elements into a vector in some canonical order, say lexicographic. We write $\boldsymbol{w} \leftarrow\!\!\$\, \mathsf{S2V}(S)$ to say that the ordering is random, meaning the entries of $\boldsymbol{w}$ are a random permutation of the elements of S. We say $\boldsymbol{w}$ is a vector over S if $\mathsf{V2S}(\boldsymbol{w}) \subseteq S$. By S^* we denote the set of all finite-length vectors over S.

Strings are identified with vectors over $\{0,1\}$, so that ε denotes the empty string, $\{0,1\}^*$ denotes the set of all finite-length strings, $|Z|$ denotes the length of a string Z and $Z[i]$ denotes its i-th bit. By $x \| y$ we denote the concatenation of strings x, y. If x, y are equal-length strings then $x \oplus y$ denotes their bitwise xor.

If X is a finite set, then $|X|$ denotes its size and $x \leftarrow\!\!\$\, X$ denote picking x uniformly at random from X. By $\mathcal{P}(X)$ we denote the power set of set X, meaning the set of all subsets of X. For integers $a \le b$ we let $[a..b]$ be shorthand for $\{a, \ldots, b\}$. We use $1, 0$ to indicate the booleans "true" and "false" respectively, and $[[B]]$ returns 1 if boolean expression B is true and 0 otherwise. We use $\perp$ (bot) as a special symbol to denote rejection, and it is assumed to not be in $\{0,1\}^*$.

We let $\mathbb{G}^* = \mathbb{G} \setminus \{1\}$ be the set of non-identity elements of a group $\mathbb{G}$. By $\langle g \rangle$ we denote the set of all powers of $g \in \mathbb{G}$, so writing $\mathbb{G} = \langle g \rangle$ indicates that g is a generator of $\mathbb{G}$. In that case, $\mathrm{dlog}_{\mathbb{G},g}(A) \in \mathbb{Z}_p$ is the discrete logarithm of $A \in \mathbb{G}$ to base g, where p is the order of $\mathbb{G}$.

ORACLE SPACES AND RANDOM ORACLES. In the random oracle model [13], the domain and range of the random oracle can depend on the scheme. (The latter term here includes protocols and functionalities.) Accordingly, we let a scheme S specify a set OS (or $\mathsf{S.OS}$ if disambiguation is needed) of functions, called the *oracle space*. The game will then pick a function $\mathsf{H} \leftarrow\!\!\$\, \mathsf{OS}$ at

random and provide as random oracle a procedure RO that when queried with X returns $\mathsf{H}(X)$. This approach is flexible. By different choices of the oracle space, one can capture other idealized models such as the ideal cipher or ideal permutation models. One can also capture a standard model instantiation of a ROM scheme, by for example setting OS to be a singleton set consisting of the $\mathsf{SHA512}$ hash function. Finally, if OS is absent or empty, one is in the standard model directly.

ALGORITHMS. Functions (we will not consider uncomputable ones) are identified with deterministic algorithms. If OS is an oracle space (i.e. a set of functions) then we write $A\colon [\mathsf{OS}]\times D_1\times\cdots\times D_n \to R$ to mean that A is an algorithm taking as oracle a function $\mathsf{H}\in\mathsf{OS}$ and taking inputs $x_1,\ldots,x_n$ with $x_i\in D_i$ for $i\in[1..n]$, to return an output $y\leftarrow\!\!\$\ A[\mathsf{H}](x_1,\ldots,x_n)\in R$. We let $\mathbf{Out}(A[\mathsf{H}](x_1,\ldots,x_n))$ denote the set of all possible outputs of A on the given inputs. Running time is worst case, which for an algorithm with access to an oracle means across all possible replies from the oracle. If we want to make A's coins (random choices) explicit we may see it as a deterministic algorithm $A\colon [\mathsf{OS}]\times D_1\times\cdots\times D_n\times\Omega\to R$ so that $y\leftarrow\!\!\$\ A[\mathsf{H}](x_1,\ldots,x_n)$ is shorthand for picking $\omega\leftarrow\!\!\$\ \Omega$ and returning $y\leftarrow A[\mathsf{H}](x_1,\ldots,x_n;\omega)$. Omitting OS and the H argument return us to the standard model.

GAMES. We use the code-based game-playing framework of [15]. A game G specifies an INITIALIZE procedure, further procedures (also called oracles) and a FINALIZE procedure. In the ROM [13], which we use throughout, the random oracle appears as a game procedure RO. When game G is executed with adversary A, first INITIALIZE executes and what it returns is the input to A. Then A runs and can call oracles other than INITIALIZE, FINALIZE. When A halts, its output is the input to FINALIZE, and the output of the latter is the game output. By $\Pr[\mathrm{G}(A)\Rightarrow y]$ we denote the probability that the execution of game G with adversary A results in the game output being y, and write just $\Pr[\mathrm{G}(A)]$ for $\Pr[\mathrm{G}(A)\Rightarrow 1]$.

Different games may have procedures (oracles) with the same names, and if we need to disambiguate, we may write G.O to refer to oracle O of game G. In game pseudocode, integer variables, set variables, boolean variables and string variables are assumed initialized, respectively, to 0, the empty set $\emptyset$, the boolean 0 and $\perp$. Adversaries in games are always assumed to be domain-respecting, meaning if a query they provide is expected to fall in some scheme-associated set, then it does. The running time of an adversary by convention is the execution time of the game with the adversary, so that the time taken by oracles to respond to adversary queries is included. We write $\mathrm{Q}^{\mathrm{O}}(A)$ to denote the number of queries made to oracle O in the execution of the game with A. Note that by convention, again, both queries made directly by A and those made by scheme algorithms are included. In particular, $\mathrm{Q}^{\mathrm{RO}}(A)$ includes the queries made by scheme algorithms either explicitly to RO or instead directly to the function H underlying RO, in the execution of the game with A. We say that adversary A_2 (playing a game G_2) has the same query profile as adversary A_1 (playing a game G_1) if the games provide oracles of the same names (even if not same behavior), and the number of queries to each of these oracles is the same for both adversaries.

For the following, recall that games G,H are identical-until-bad if their code differs only in statements that follow the setting of flag bad to 1 [15].

Lemma 2.1 [Fundamental Lemma of Game Playing [15]] *Let* G,H *be identical-until-*bad *games. Then for any adversary* A *we have*

$$|\Pr[\mathrm{G}(A)]-\Pr[\mathrm{H}(A)]|\le\Pr[\mathrm{H}(A)\text{ sets }\mathsf{bad}]=\Pr[\mathrm{G}(A)\text{ sets }\mathsf{bad}]\ .$$

CONCRETE SECURITY. In this setting, there is no explicit security parameter, and thus no *formal* definition of either polynomial time (for an adversary) or negligible (for its advantage). We simply define advantage functions, and theorems relate them with explicit bounds. Discussions will still informally use terms like polynomial-time or negligible with the natural interpretations.

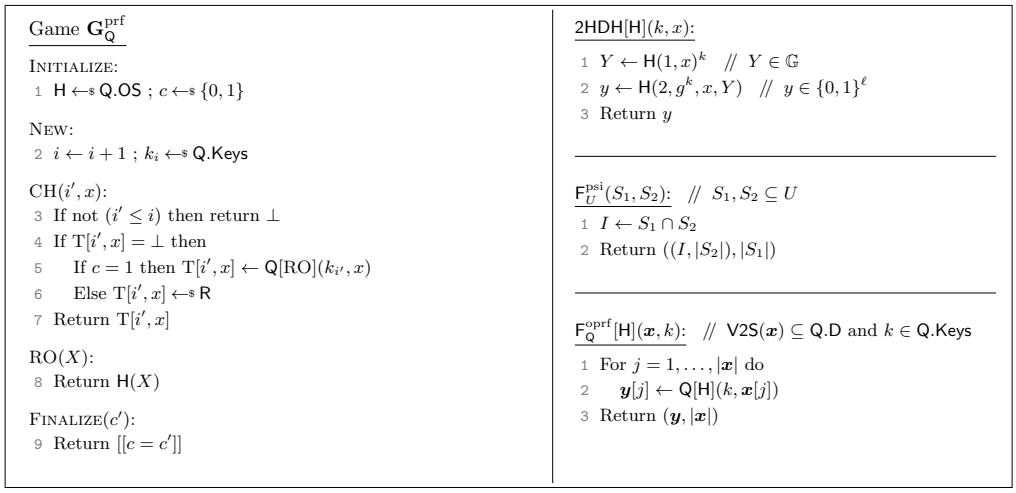


Figure 3: **Left:** PRF game for function family Q. **Right:** On the top is the 2HDH function family associated to group $\mathbb{G} = \langle g \rangle$ and integer ℓ, and, below it, the PSI functionality over universe U. At the bottom is the OPRF functionality associated to PRF Q.

SECURITY OF FUNCTION FAMILIES. A family of functions $\mathsf{Q}\colon [\mathsf{OS}] \times \mathsf{Keys} \times \mathsf{D} \to \mathsf{R}$ takes a key $K \in \mathsf{Keys}$ and input $X \in \mathsf{D}$ and, with oracle access to $\mathsf{H} \in \mathsf{OS}$, returns an output $Y \leftarrow \mathsf{Q}[\mathsf{H}](K, X)$. For emphasis or disambiguation, we may write $\mathsf{Q.OS}, \mathsf{Q.Keys}, \mathsf{Q.D}, \mathsf{Q.R}$ for the different subcomponents of Q.

A security metric for Q that we will use is PRF security [29] in the multi-user setting [7]. The prf (pseudorandom function) advantage of adversary A_{prf} is defined as $\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{Q}}(A_{\mathrm{prf}}) = 2\Pr[\mathbf{G}^{\mathrm{prf}}_{\mathsf{Q}}(A_{\mathrm{prf}})] - 1$ where the game is on the left in Figure 3.

As an example, the top right of Figure 3 shows the 2H-DH function family $\mathsf{2HDH}\colon [\mathsf{OS}] \times \mathbb{Z}_p \times \{0,1\}^* \to \{0,1\}^\ell$ underlying the 2H-DH OPRF [36]. It is associated to a group $\mathbb{G} = \langle g \rangle$ of prime order p generated by $g \in \mathbb{G}$, and an integer $\ell \ge 1$. Here OS is the set of all functions H such that $\mathsf{H}(1, \cdot)\colon \{0,1\}^* \to \mathbb{G}$ and $\mathsf{H}(2, \cdot, \cdot, \cdot)\colon \mathbb{G} \times \{0,1\}^* \times \mathbb{G} \to \{0,1\}^\ell$. This function family conceptually uses two random oracles $\mathsf{H}(1, \cdot), \mathsf{H}(2, \cdot, \cdot, \cdot)$ that are packaged into one to respect our formalism. The following says that 2HDH is PRF-secure in the ROM.

Proposition 2.2 *Let $\mathbb{G} = \langle g \rangle$ be a group of prime order p, and $\ell \ge 1$ an integer. Let* 2HDH *be the associated 2H-DH family of functions as per Figure 3. Let A_{prf} be an adversary playing game $\mathbf{G}^{\mathrm{prf}}_{\mathsf{2HDH}}$. Then*

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{2HDH}}(A_{\mathrm{prf}}) \le \frac{(\mathrm{Q}^{\mathrm{RO}}(A_{\mathrm{prf}}) + \mathrm{Q}^{\mathrm{NEW}}(A_{\mathrm{prf}})) \cdot \mathrm{Q}^{\mathrm{NEW}}(A_{\mathrm{prf}})}{p} .$$

We omit a formal proof, but the intuition is that, when the challenge bit is 1, outputs of the challenge oracle are distributed uniformly in R as long as a certain "bad" event does not happen, the event being either a collision in keys across NEW queries, or the random oracle being queried on $g^{k_{i'}}$ for a $k_{i'}$ picked by NEW. Thus it suffices to bound the probability of this bad event.

In Section 4, we show that server-side security of any OPRF implies PRF security of the family of functions underlying the OPRF.

3 2PC Definitional Framework

We give our core definitions of syntax and security in a concrete setting, and then turn to relations between definitions. We see the party identities as $1, 2$ with 1 being the "client" and 2 being the "server."

3.1 Core definitions

FUNCTIONALITIES. A (two-party) functionality describes the function that the parties want to compute. Formally, it is an algorithm $\mathsf{F}\colon [\mathsf{OS}] \times \mathsf{D}_1 \times \mathsf{D}_2 \to \mathsf{R}_1 \times \mathsf{R}_2$. The functionalities of practical interest that we want to treat are deterministic, so for simplicity we restrict attention in this work to deterministic F, and this is assumed moving forward. We leave treatment of randomized functionalities to future work. Now, to explain, given as oracle $\mathsf{H} \in \mathsf{OS}$, and inputs $x_1 \in \mathsf{D}_1$ and $x_2 \in \mathsf{D}_2$ of parties 1,2 respectively, the functionality returns outputs $y_1 \in \mathsf{R}_1$ and $y_2 \in \mathsf{R}_2$ for parties 1,2, respectively, via $(y_1, y_2) \leftarrow \mathsf{F}[\mathsf{H}](x_1, x_2)$.

Allowing F to have access to a random oracle is important to capture some OPRFs. As per our vector notation, for $i \in \{1, 2\}$ we may write $\mathsf{F}[\mathsf{H}](x_1, x_2)[i]$ for the i-th component of the 2-vector $\mathsf{F}[\mathsf{H}](x_1, x_2)$.

PSI AND OPRF FUNCTIONALITIES. The right side of Figure 3 shows two examples. First, the PSI functionality $\mathsf{F}^{\mathrm{psi}}_U\colon \mathcal{P}(U) \times \mathcal{P}(U) \to (\mathcal{P}(U) \times \mathbb{N}) \times \mathbb{N}$ is associated to a set U called the universe. This functionality does not use a random oracle. Party $i \in \{1, 2\}$ has input a set $S_i \subseteq U$. The intersection I of the two sets is returned to party 1, and both parties are also given set-size information because protocols tend to leak it.

Second, let $\mathsf{Q}\colon [\mathsf{OS}] \times \mathsf{Keys} \times \mathsf{D} \to \mathsf{R}$ be a family of functions. We associate to it the OPRF functionality $\mathsf{F}^{\mathrm{oprf}}_{\mathsf{Q}}\colon [\mathsf{OS}] \times \mathsf{D}^* \times \mathsf{Keys} \to \mathsf{R}^* \times \mathbb{N}$. The input of the server (party 2) is a PRF key $k \in \mathsf{Keys}$. The input of the client (party 1) is vector $\boldsymbol{x}$ over D. The functionality computes a corresponding vector $\boldsymbol{y}$, over R, of outputs under $\mathsf{Q}[\mathsf{H}](k, \cdot)$, that goes to the client. The server gets the length of $\boldsymbol{x}$ since protocols tend to leak it. In particular if $\mathsf{Q} = \mathsf{2HDH}$ is the 2HDH PRF of Figure 3 then $\mathsf{F}^{\mathrm{oprf}}_{\mathsf{Q}}$ is the 2H-DH OPRF functionality, protocols for which we will analyze. Note our definition extends the usual ones by allowing the client input to be a vector over D rather than a single point in D.

PROTOCOLS. Party $i \in \{1, 2\}$ has input x_i. The parties now use an interactive protocol to interact towards computing outputs for some target functionality. But what exactly (meaning, mathematically or definitionally) is a protocol? In UC [19] and Goldreich's textbooks [27, 28], it is a pair of interactive TMs. In some parts of the literature (including Lindell's tutorial [43]) it is not formalized at all. We will give a usable yet rigorous formalization of a protocol as an algorithm that takes a current state and an incoming message to return an updated state and outgoing message.

Thus, formally, a *protocol* Π is an algorithm $\Pi\colon [\mathsf{OS}] \times \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$. It may be randomized, and $\Pi\colon [\mathsf{OS}] \times \{0,1\}^* \times \{0,1\}^* \times \Omega \to \{0,1\}^* \times \{0,1\}^*$ denotes the underlying deterministic algorithm with Ω the set of coins. As a function of its current state $st \in \{0,1\}^*$, a received message $m_{\mathrm{in}} \in \{0,1\}^*$ and its coins $\omega \in \Omega$, a party computes its outgoing message m_{out} as well as, for itself, an updated state st, written $(st, m_{\mathrm{out}}) \leftarrow \Pi[\mathsf{H}](st, m_{\mathrm{in}}; \omega)$. As usual, we write $(st, m_{\mathrm{out}}) \leftarrow\!\!\$\ \Pi[\mathsf{H}](st, m_{\mathrm{in}})$ for picking $\omega \leftarrow\!\!\$\ \Omega$ and letting $(st, m_{\mathrm{out}}) \leftarrow \Pi[\mathsf{H}](st, m_{\mathrm{in}}; \omega)$. A party's state records its input as $st.\mathrm{in}$, its output as $st.\mathrm{out}$ and its decision to accept or reject as $st.\mathrm{dec} \in \{1, 0\}$. The interaction consists of $\mathsf{nr} \in \mathbb{N}$ moves (also called rounds). The convention is that party 1 sends the first message.

EXECUTION TRACES. A protocol may be (honestly) executed on inputs x_1, x_2, coins $\omega_1, \omega_2 \in \Omega$ for

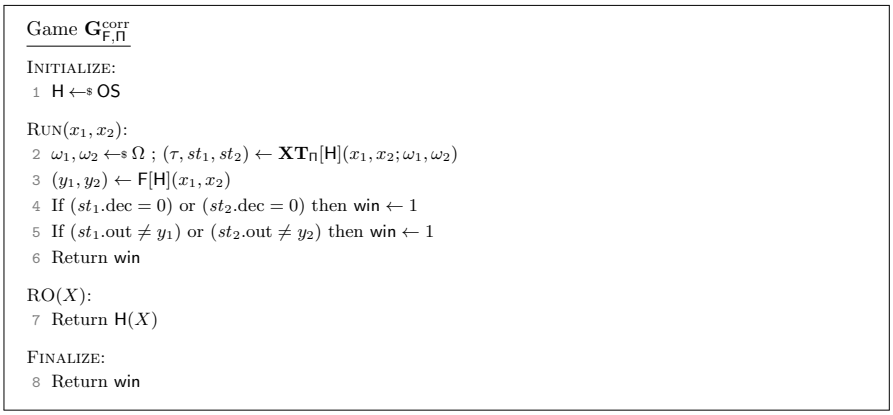


Figure 4: Game assessing correctness of protocol Π for functionality F.

---

the parties and access to an oracle $\mathsf{H} \in \mathsf{OS}$ to generate an execution trace $(\tau, st_1, st_2) \leftarrow \mathbf{XT}_{\Pi}[\mathsf{H}](x_1, x_2; \omega_1, \omega_2)$. Here τ is a transcript of the interaction, which is the sequence of messages exchanged, and st_1, st_2 are the final states of the parties. In detail:

$\underline{\mathbf{XT}_{\Pi}[\mathsf{H}](x_1, x_2; \omega_1, \omega_2)}$
$st_1.\mathrm{in} \leftarrow x_1$; $st_2.\mathrm{in} \leftarrow x_2$; $m_0 \leftarrow \varepsilon$; $i \leftarrow 1$
For $j = 1, \ldots, \mathsf{nr}$ do
  $(st_i, m_j) \leftarrow \Pi[\mathsf{H}](st_i, m_{j-1}; \omega_i)$; $i \leftarrow 3 - i$
$\tau \leftarrow (m_1, \ldots, m_{\mathsf{nr}})$; Return (τ, st_1, st_2)

As indicated above, the outputs and decisions can be recovered from the final states of the parties.

Correctness. Correctness asks that an honest execution of a protocol computes the target functionality. This is straightforward enough to define for perfect correctness, but we need a clear definition of imperfect correctness that in particular allows quantifying correctness failure in protocols where it depends on computational assumptions. Accordingly, we treat correctness in detail, using a game.

Let $\mathsf{F}\colon [\mathsf{OS}] \times \mathsf{D}_1 \times \mathsf{D}_2 \to \mathsf{R}_1 \times \mathsf{R}_2$ be a functionality and Π a protocol. We assume for simplicity that the functionality and protocol have the same oracle space, which is wlog. Define the correctness advantage of adversary A_{corr} as $\mathbf{Adv}^{\mathrm{corr}}_{\mathsf{F},\Pi}(A_{\mathrm{corr}}) = \Pr[\mathbf{G}^{\mathrm{corr}}_{\mathsf{F},\Pi}(A_{\mathrm{corr}})]$ where the game is in Figure 4. Here the adversary can run the protocol on inputs (x_1, x_2) of its choice by calling oracle Run. It wins if either the parties reject or their outputs do not match those of the functionality. Note that multiple calls to Run are allowed. We say Π is perfectly correct for F if $\mathbf{Adv}^{\mathrm{corr}}_{\mathsf{F},\Pi}(A_{\mathrm{corr}}) = 0$ for all A_{corr}, regardless of the running time and number of oracle queries of A_{corr}. But having defined this advantage function allows us to make clear and precise statements about imperfect correctness. This will allow us to see how the correctness advantage grows with the number of oracle queries in PSI protocols where correctness depends on computational assumptions.

Security. We will be considering security in the *semi-honest* or *honest-but-curious* model where it is assumed that the corrupt party does not deviate from the protocol but, at the end, given its view (conversation transcript and its own coins) tries to find information about the other party's input. By convention, we will refer to this other party as the honest one.

We start with a new indistinguishability-style definition that we call *input indistinguishability* (InI). This is a single-quantifier definition whose first merit is simplicity relative to the usual simulation-style (double-quantifier) definitions. Additionally, as discussed in the Introduction, it is "concrete-security friendly," meaning allows one to show concrete bounds on adversary advantage

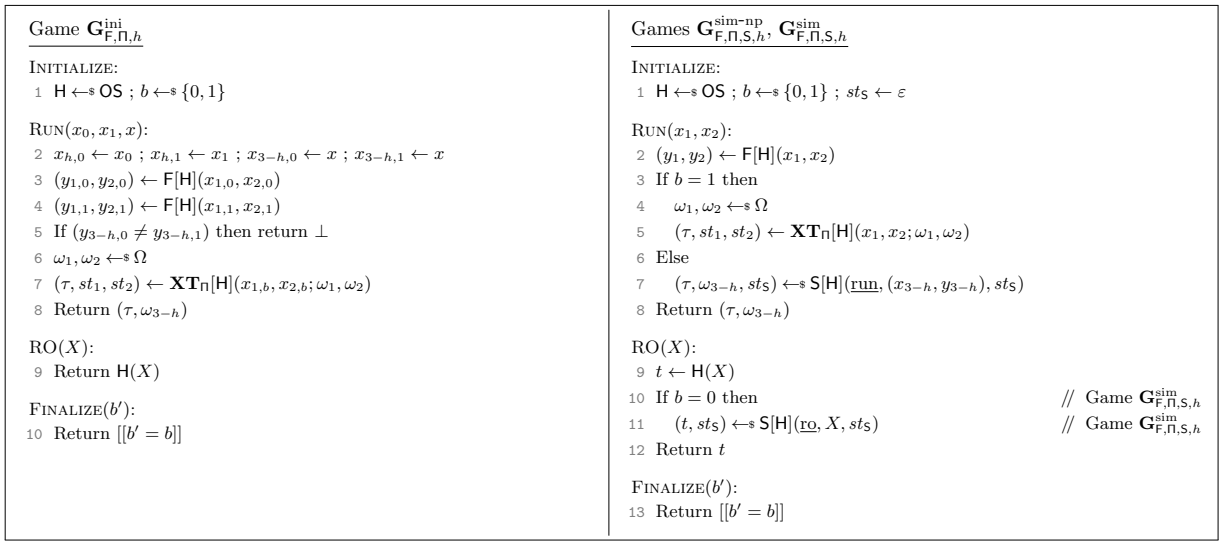


Figure 5: **Left:** Game defining InI security for protocol Π for functionality F, where $h \in \{1,2\}$ is the honest party. **Right:** Games defining SIM-np (lines 10-11 excluded) and SIM (lines 10-11 included) security for protocol Π for a functionality F, where $h \in \{1,2\}$ is the honest party and S is the simulator.

from which parameters providing a desired level of security may be easily determined. Following the tradition in this area [43, 19], we will also formulate (double-quantifier) simulation-based definitions. Our definitions explicitly and flexibly incorporate the ROM, based on which our simulation-based definition comes in two forms: in the first, SIM, the random oracle is programmable by the simulator, and in the second, stronger definition SIM-np, it is not.

In our definitions, an adversary triggers a protocol execution, on inputs of its choice, via a query to an oracle Run. We allow multiple queries to Run, to capture the real-life expectation of multiple executions of the protocol on different inputs. This allows us to measure (and then reduce) the degradation of security as a function of the number of Run calls.

We will give a complete picture of the relations between our three definitions. The main important takeaway is that InI is not weaker than the simulation-based definitions, but equivalent to SIM-np for functionalities satisfying a condition we define and call *invertibility*. We show that it is met by many natural and practical functionalities including PSI, so that, for results, we can then focus on InI.

For all the following definitions, we let $\mathsf{F}\colon [\mathsf{OS}] \times \mathsf{D}_1 \times \mathsf{D}_2 \to \mathsf{R}_1 \times \mathsf{R}_2$ be the functionality. We let Π be a protocol for it with $\Pi\colon [\mathsf{OS}] \times \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$. We assume wlog that the RO spaces of F, Π are the same. (One can always work with an appropriate union of the two spaces if not.)

InI definition. Input-indistinguishability (InI) is defined via game $\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,h}$ in Figure 5. The ini-advantage of an adversary A_{ini} is then defined by $\mathbf{Adv}^{\text{ini}}_{\mathsf{F},\Pi,h}(A_{\text{ini}}) = 2\Pr[\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,h}(A_{\text{ini}})] - 1$. To explain, here $h \in \{1,2\}$ is the "honest" party, meaning the adversary is playing the role of party $3-h$ and trying to learn something about the honest party's input. The adversary can query Run with two choices $x_0, x_1 \in \mathsf{D}_h$ of inputs for the honest party and a single choice $x \in \mathsf{D}_{3-h}$ for the corrupt party. This results in two pairs of inputs for the functionality. At lines 3-4 the functionality is evaluated on both pairs. If the resulting outputs $y_{3-h,0}$ and $y_{3-h,1}$ for the corrupt party differ, then the game returns $\perp$ at line 5 to avoid trivial distinguishing. Else, the protocol is run with the honest-party input being determined by the challenge bit b from line 1, and the resulting conversation transcript and the corrupt party's coins ω_{3-h} are returned to the adversary.

Multiple queries to RUN are allowed.

Asymptotically we would say that Π is InI secure for F if for every PPT $A_{\rm ini}$ the function $\mathbf{Adv}^{\rm ini}_{\mathsf{F},\Pi,h}(A_{\rm ini})$ is negligible, which illustrates how this is a single-quantifier definition. As usual in the concrete setting there is no formal definition of being "secure;" we give only a formal metric of security and our results in this concrete setting will relate advantages.

SIM-NP DEFINITION. Moving to our simulation-based definitions, we start with SIM-np, the non-programmable ROM one. It is given via game $\mathbf{G}^{\text{sim-np}}_{\mathsf{F},\Pi,\mathsf{S},h}$ in Figure 5, where S is an algorithm called the simulator. As before, the game is also parameterized by the identity $h \in \{1,2\}$ of the honest party, whose input the adversary $A_{\rm snp}$, in the role of the corrupted party $3-h$, is trying to learn. Lines 10-11 are not present in this game. Line 1 picks a random challenge bit b. If $b = 1$ then we have the "real" game and if $b = 0$ the "ideal" game. The adversary can call RUN, giving it inputs for both parties. In response it obtains a conversation transcript τ, and coins ω_{3-h} for the corrupted party. It can query this oracle as often as it wants. In the real game, the transcript and coins are determined by running the protocol, while in the ideal game, they are determined by the simulator. Queries to the random oracle RO are answered via $\mathsf{H} \in \mathsf{OS}$. The first argument to the simulator is a keyword indicating the role in which it is being run, and st_{S} is its state. The latter is initialized at line 1. After that, when the simulator runs (line 7) it takes its current state and returns an updated state. The state variable st_{S} is maintained by the game. The non-programmability of the RO is in the fact that the RO oracle simply responds via H and the simulator gets access to the same H. We let $\mathbf{Adv}^{\text{sim-np}}_{\mathsf{F},\Pi,\mathsf{S},h}(A_{\rm snp}) = 2\Pr[\mathbf{G}^{\text{sim-np}}_{\mathsf{F},\Pi,\mathsf{S},h}(A_{\rm snp})] - 1$ be the advantage of an adversary $A_{\rm snp}$.

In an asymptotic setting, we would say that Π is SIM-np secure for F and h if there is a PPT S such that for every PPT $A_{\rm snp}$ the function $\mathbf{Adv}^{\text{sim-np}}_{\mathsf{F},\Pi,\mathsf{S},h}(A_{\rm snp})$ is negligible. This illustrates how this is a double-quantifier definition. As usual, in our concrete setting, theorems (e.g. Theorem 3.2) will relate advantages.

SIM DEFINITION. The programmable ROM version of our simulation-based definition of security, called SIM, is given via game $\mathbf{G}^{\text{sim}}_{\mathsf{F},\Pi,\mathsf{S},h}$ in Figure 5. As before, $h \in \{1,2\}$ is the identity of the honest party and S is the simulator. Lines 10–11 (now included and the only change from SIM-np) represent the programming, allowing the simulator to determine the output of oracle RO. We continue to give the simulator access to an actual random oracle via H, which it can use or ignore as it wishes. As we will explain below, it is important for the meaningfulness of this definition that the functionality queries to the random oracle at line 2 are not programmed by, or even visible to, the simulator. We expect that st_{S} holds the current input-output table of the simulated random oracle, and whatever RO answers the simulator may need for the lines 7,11 simulations, it can create and store in st_{S} if they do not already exist there. An adversary $A_{\rm sim}$ again has to find the correct value of the challenge bit b to win. We let $\mathbf{Adv}^{\text{sim}}_{\mathsf{F},\Pi,\mathsf{S},h}(A_{\rm sim}) = 2\Pr[\mathbf{G}^{\text{sim}}_{\mathsf{F},\Pi,\mathsf{S},h}(A_{\rm sim})] - 1$ be its advantage.

Again, in an asymptotic setting, we would say that Π is SIM-secure for F and h if there is a PPT S such that for every PPT $A_{\rm sim}$ the function $\mathbf{Adv}^{\text{sim}}_{\mathsf{F},\Pi,\mathsf{S},h}(A_{\rm sim})$ is negligible.

A SUBTLE POINT ABOUT SIM. At line 2 in game $\mathbf{G}^{\text{sim}}_{\mathsf{F},\Pi,\mathsf{S},h}$ (right panel of Figure 5), RO queries of the functionality F, if any, are answered by an honest random function H. This may not be the first or obvious choice; why not have these also be answered by the simulator like the answers to other RO queries in this game? To explain, let us denote by SIM* the variant we have just mentioned, namely it is the same as SIM except that, at line 2, we replace $\mathsf{F}[\mathsf{H}](x_1,x_2)$ with $\mathsf{F}[\mathrm{RO}](x_1,x_2)$, so that, when $b = 0$, the RO queries of F are answered by the simulator at line 11. A self-contained and formal definition of SIM*, as well as a more precise and formal rendition of what follows, is in Appendix A.

We claim that SIM* is an incorrect (unsound) definition for functionalities F that access the RO. (If F does not access RO, there is no difference between SIM and SIM*, and both are sound.) Specifically, our claim is that, if F can access the RO, then obviously insecure protocols can be shown secure under SIM*.

As an example, let $\mathsf{F} = \mathsf{F}^{\mathrm{oprf}}_{\mathsf{2HDH}}$ be the OPRF functionality (Figure 3) associated to the 2H-DH PRF $\mathsf{2HDH}\colon [\mathsf{OS}] \times \mathbb{Z}_p \times \{0,1\}^* \to \{0,1\}^\ell$ (Figure 3). Recall that here OS is the set of all functions H such that $\mathsf{H}(1,\cdot)\colon \{0,1\}^* \to \mathbb{G}$ and $\mathsf{H}(2,\cdot,\cdot,\cdot)\colon \mathbb{G} \times \{0,1\}^* \times \mathbb{G} \to \{0,1\}^\ell$. Suppose party 1 (client) has input $x \in \{0,1\}^*$ —formally, the 1-vector (x)— while party 2 (server) has input a key $k \in \mathbb{Z}_p$. Consider the following protocol Π: (1) Party 1 sends its entire input x to party 2 (2) party 2 computes $Y \leftarrow \mathsf{H}(1,x)^k$, sends (Y, g^k) to party 1, and outputs 1 as its own output, and finally (3) party 1 computes and outputs $y \leftarrow \mathsf{H}(2, g^k, x, Y)$.

This protocol should clearly be considered insecure for honest party $h = 1$ since from the conversation transcript an adversary learns the entire input x of party 1, which it cannot deduce given just the functionality output (namely 1) for the corrupted party (namely party 2). Yet, it is possible to design a successful simulator for Π under SIM*. Why? At line 2 on the right of Figure 5, F would query $X = (1,x)$ to RO to compute $Y \leftarrow \mathsf{H}(1,x)^k$. But this query X is passed at line 11 to the simulator, who thus directly learns x. It can store x in its state, and can now easily produce the transcript τ at line 7. Namely, it knows the input k of the corrupted party and can thus compute $Y \leftarrow \mathsf{H}(1,x)^k$ and return $(x, (Y, g^k))$ as the transcript. So this protocol is SIM* secure despite being intuitively insecure. This anomaly goes away with SIM, where now the query $1, x$ made to H at line 2 is not visible to the simulator.

3.2 Relations between definitions

Simulation-based definitions have been the paradigm in the 2PC area, are well accepted and provide intuitively strong security, but their double-quantifier nature increases complexity and reduces concrete-security friendliness. Our single-quantifier InI definition is in contrast simple and concrete-security friendly, but one must ask if this is at the cost of strength, meaning have we lowered security relative to the SIM and SIM-np definitions? The main result of this section (Theorem 3.4) says that usually not: for functionalities satisfying a condition we define, that we call invertibility and show is met by functionalities of practical interest, InI is just as strong as SIM or SIM-np. In this way, it provides the "best of both worlds."

This result emerges as part of a comprehensive picture of relations between the notions that was summarized in Figure 1. In this section, we give the formal statements and proofs for the results in that figure, culminating with Theorem 3.4. All our results are in the concrete-security setting.

SIM-NP IMPLIES SIM. We start by confirming that SIM-np implies SIM, meaning the non-programmable ROM definition is stronger than the programmable one. Fix a functionality F and protocol Π for it. Now, SIM-np security implies there is a SIM-np-simulator S for game $\mathbf{G}^{\text{sim-np}}_{\mathsf{F},\Pi,\mathsf{S},h}$. This S provides a subroutine corresponding to the run role. We want to construct a SIM-simulator for game $\mathbf{G}^{\text{sim}}_{\mathsf{F},\Pi,\mathsf{S},h}$. We do this by making it the same as S except we add a subroutine for the ro role that, given a query X to the RO, just returns $\mathsf{H}(X)$, where H is the RO provided to S. The advantage of an adversary A across the games is preserved. The following formalizes the result. The proof is simple and is omitted.

Theorem 3.1 [SIM-np ⇒ SIM] *Let F be a functionality and Π a protocol for it. Let $h \in \{1,2\}$ be the honest party. Given a simulator S defining $\mathsf{S}[\cdot](\underline{\text{run}},\cdot,\cdot)$, extend it to also define $\mathsf{S}[\cdot](\underline{\text{ro}},\cdot,\cdot)$ by*

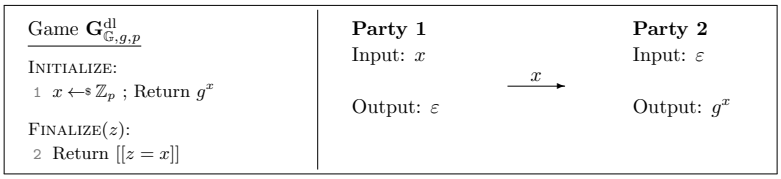


Figure 6: **Left:** Discrete log game for group $\mathbb{G} = \langle g \rangle$ of order p. **Right:** Protocol Π for Theorem 3.3.

$\mathsf{S}[\mathsf{H}](\underline{\mathrm{ro}}, X, st_{\mathsf{S}}) = (\mathsf{H}(X), st_{\mathsf{S}})$. *Then for any adversary A we have*

$$\mathbf{Adv}^{\mathrm{sim}}_{\mathsf{F},\Pi,\mathsf{S},h}(A) = \mathbf{Adv}^{\text{sim-np}}_{\mathsf{F},\Pi,\mathsf{S},h}(A)\ . \tag{1}$$

How does this statement show that SIM-np implies SIM? Assume Π is SIM-np-secure for F. Then there is a PPT SIM-np-simulator S such that $\mathbf{Adv}^{\text{sim-np}}_{\mathsf{F},\Pi,\mathsf{S},h}(A)$ is negligible for all PPT A. The extended S defined by the theorem is a PPT SIM-simulator, and Eq. (1) implies that $\mathbf{Adv}^{\mathrm{sim}}_{\mathsf{F},\Pi,\mathsf{S},h}(A)$ is also negligible for all PPT A, and Π is thus SIM-secure. In terms of reductions, Eq. (1) represents a trivial one which maps A to itself.

SIM implies InI. The following says that SIM always implies InI. That is, if Π is SIM secure for F then Π is also InI secure for F. The proof is in Appendix B.

Theorem 3.2 [SIM $\Rightarrow$ InI] *Let F be a functionality and Π a protocol for it. Let $h \in \{1, 2\}$ be the honest party. Let A_{ini} be an adversary playing game $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F},\Pi,h}$. Then we can construct an adversary A_{sim} such that for all simulators S we have*

$$\mathbf{Adv}^{\mathrm{ini}}_{\mathsf{F},\Pi,h}(A_{\mathrm{ini}}) \le 2 \cdot \mathbf{Adv}^{\mathrm{sim}}_{\mathsf{F},\Pi,\mathsf{S},h}(A_{\mathrm{sim}})\ . \tag{2}$$

Adversary A_{sim} has the same query profile as A_{ini} and about the same running time.

It may seem strange that Eq. (2) holds for *all* simulators. In particular, how does this show that SIM implies InI? The answer is that if Π is SIM-secure for F then there is a particular, PPT simulator S such that $\mathbf{Adv}^{\mathrm{sim}}_{\mathsf{F},\Pi,\mathsf{S},h}(A_{\mathrm{sim}})$ is negligible. Now by using S in Eq. (2) we can conclude that $\mathbf{Adv}^{\mathrm{ind}}_{\mathsf{F},\Pi,h}(A_{\mathrm{ini}})$ is also negligible, meaning Π is InI-secure for F.

InI does not always imply SIM. We will eventually show that InI implies not only SIM, but even SIM-np, for a large class of important functionalities. But first we want to caution and clarify that it does not do so for all functionalities. This is done via a counterexample. That this counterexample is contrived and artificial only reinforces our view that the implication will hold for natural functionalities.

Specifically we now give a functionality F, and a protocol Π for it, such that Π is InI-secure but not SIM-secure. We assume for this the hardness of the discrete logarithm problem in a cyclic group $\mathbb{G}$. The formalization for the latter is via the DL game $\mathbf{G}^{\mathrm{dl}}_{\mathbb{G},g,p}$ shown in the left panel of Figure 6. It is associated to group $\mathbb{G} = \langle g \rangle$ of order p with generator $g \in \mathbb{G}$. The advantage of an adversary A_{dl} playing this game is given by $\mathbf{Adv}^{\mathrm{dl}}_{\mathbb{G},g,p}(A_{\mathrm{dl}}) = \Pr[\mathbf{G}^{\mathrm{dl}}_{\mathbb{G},g,p}(A_{\mathrm{dl}})]$. The proof of the following is in Appendix C.

Theorem 3.3 [InI $\not\Rightarrow$ SIM in general] *Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of order p. Let $\mathsf{F} : \mathbb{Z}_p \times \{\varepsilon\} \to \{\varepsilon\} \times \mathbb{G}$ be the functionality defined by $\mathsf{F}(x, \varepsilon) = (\varepsilon, g^x)$ for all $x \in \mathbb{Z}_p$. Let Π be the protocol for F shown in Figure 6. Then:*

1. *Π is ini-secure for F: For any adversary A_{ini} playing game $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F},\Pi,1}$, we have*

$$\mathbf{Adv}^{\mathrm{ini}}_{\mathsf{F},\Pi,1}(A_{\mathrm{ini}}) = 0\ . \tag{3}$$

2. Π *is sim-insecure for* F *assuming the DL problem is hard: For all simulators* S*, there exist adversaries* $A_{\rm sim}, A_{\rm dl}$*, playing games* $\mathbf{G}^{\rm sim}_{\mathsf{F},\Pi,\mathsf{S},1}$ *and* $\mathbf{G}^{\rm dl}_{\mathbb{G},g,p}$*, respectively, such that*

$$\mathbf{Adv}^{\rm sim}_{\mathsf{F},\Pi,\mathsf{S},1}(A_{\rm sim}) = 1 - \mathbf{Adv}^{\rm dl}_{\mathbb{G},g,p}(A_{\rm dl})\ . \tag{4}$$

Adversary $A_{\rm dl}$ *has the same running time as an execution of* S *in its* run *role, and* $A_{\rm sim}$ *runs in constant time.*

Why does Eq. (4) mean that Π is not SIM-secure? Let S be any PPT simulator. Then the Theorem gives PPT adversaries $A_{\rm sim}, A_{\rm dl}$ such that Eq. (4) holds. But assuming DL is hard, $\mathbf{Adv}^{\rm dl}_{\mathbb{G},g,p}(A_{\rm dl})$ is negligible, so the equation is saying that $A_{\rm sim}$ has a high (close to 1) advantage, which shows that Π is not SIM-secure for S. Since S was arbitrary, Π is not SIM-secure.

Below we show that if functionalities satisfy a condition which we call invertibility, then InI $\Rightarrow$ SIM-np. This means, using Theorem 3.1 and Theorem 3.2, that InI is equivalent to SIM for such functionalities.

Invertibility. We define *invertibility* for functionalities with respect to the honest-party identity $h \in \{1,2\}$. Let $\mathsf{F} : [\mathsf{OS}] \times \mathsf{D}_1 \times \mathsf{D}_2 \to \mathsf{R}_1 \times \mathsf{R}_2$ be a functionality. An algorithm $\mathrm{IA} : [\mathsf{OS}] \times \mathsf{D}_{3-h} \times \mathsf{R}_{3-h} \to \mathsf{D}_h$ is called an *inverter* for F and h if for all $\mathsf{H} \in \mathsf{OS}$ and all $(x_1, x_2) \in \mathsf{D}_1 \times \mathsf{D}_2$, the following always returns 1:

$(y_1, y_2) \leftarrow \mathsf{F}[\mathsf{H}](x_1, x_2)$ // Get functionality outputs
$x'_h \leftarrow\!\!\$\ \mathrm{IA}[\mathsf{H}](x_{3-h}, y_{3-h})$ // Resample an input for honest party
$x'_{3-h} \leftarrow x_{3-h}$ // Input unchanged for corrupt party
$(y'_1, y'_2) \leftarrow \mathsf{F}[\mathsf{H}](x'_1, x'_2)$ // Get new functionality outputs
Return $[[y'_{3-h} = y_{3-h}]]$ // Require corrupted-party output to be unchanged

Intuitively, consider an entity (this will be the simulator in our usage) who has an input x_{3-h} for the corrupted party. It also has an output y_{3-h} for the corrupted party, obtained from x_{3-h} and some (unknown to this entity) input x_h for the honest party. Invertibility asks that, given these, it is possible for our entity to efficiently find an input x'_h for the honest party that "explains" the output obtained by the corrupted party. It need not be that $x'_h = x_h$, and similarly need not be that $y'_h = y_h$.

In an asymptotic setting, we would say that a functionality F is *invertible* for h if there exists a PPT inverter IA for F and h. In our concrete setting, we will include the running time of IA in results.

We note that invertibility is an assumption on a functionality, not on a protocol. We also note that the functionality of Theorem 3.3 is *not* invertible for honest party 1. Indeed, inverting it would amount to solving the discrete-logarithm problem. We will see later, however, that practical functionalities including PSI are invertible.

InI implies SIM-np for invertible functionalities. Let F be a functionality that is invertible for $h \in \{1,2\}$. We show that any protocol that is InI secure for h is SIM-np secure (and thus by Theorem 3.1 also SIM secure) for h. This is done by exhibiting a simulator S under which the sim-np-advantage of any adversary $A_{\rm snp}$ can be shown small by bounding it via the ini-advantage of another adversary $A_{\rm ini}$. The assumed inverter IA will be used and run by the simulator, and then also by $A_{\rm ini}$. The following formalizes this claim. The proof is in Appendix D.

Theorem 3.4 [InI $\Rightarrow$ SIM-np for invertible functionalities] *Let* $h \in \{1,2\}$ *be the honest party. Let* F *be a functionality which is invertible for* h*, using inverter* IA*. Let* Π *be a protocol for* F*. Then*

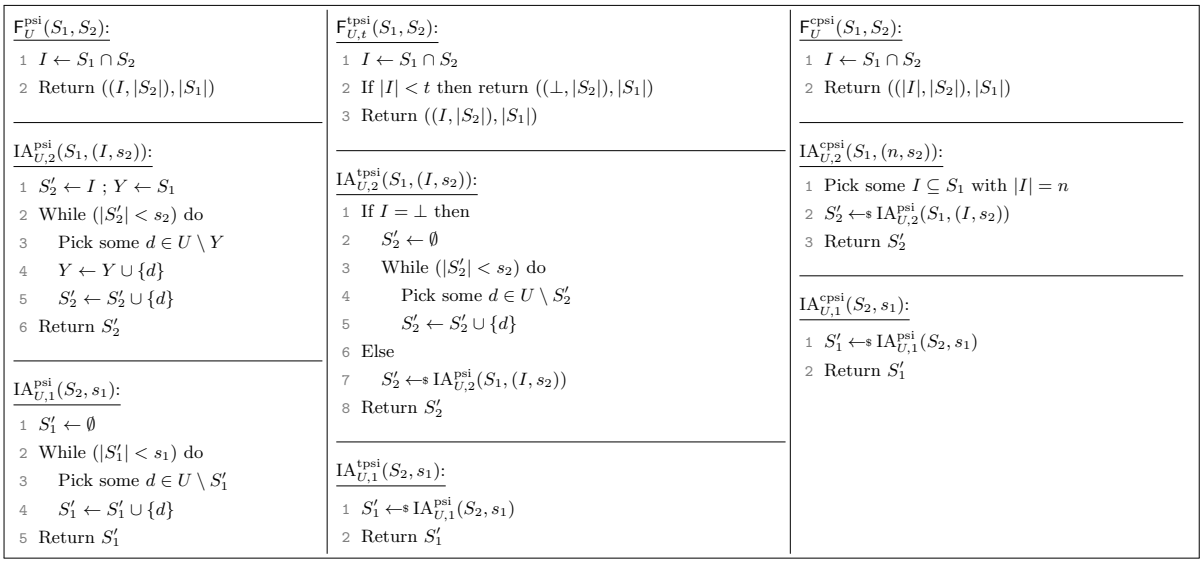


Figure 7: **Left:** PSI functionality and its inverters. **Middle:** tPSI functionality and its inverters. **Right:** cPSI functionality and its inverters.

there is a simulator S *such that the following is true. Let* A_{snp} *be any adversary playing game* $\mathbf{G}^{\text{sim-np}}_{\mathsf{F},\Pi,\mathsf{S},h}$*. Then we can construct an adversary* A_{ini} *playing game* $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F},\Pi,h}$ *such that*

$$\mathbf{Adv}^{\text{sim-np}}_{\mathsf{F},\Pi,\mathsf{S},h}(A_{\mathrm{snp}}) \leq \mathbf{Adv}^{\mathrm{ini}}_{\mathsf{F},\Pi,h}(A_{\mathrm{ini}})\ . \tag{5}$$

Adversary A_{ini} *has the same query profile as* A_{snp}*. Its running time is about that of* A_{snp}*. The running time of* S *is that of* Π *plus the time for an execution of* IA*.*

3.3 Invertibility of PSI and friends

Theorem 3.4 says that InI is just as strong as SIM-np as long as the functionality is invertible. Here we show that PSI [26], as well as a collection of PSI-related functionalities, are all invertible. This means that, for these functionalities, we can target InI without loss of security compared to simulation-based definitions, gaining in this way from the simplicity and concrete-security friendliness that the former offers compared to the latter. We show invertibility for some more functionalities in Appendix E.

Extending beyond these examples, we believe that natural functionalities of practical interest will be invertible. We clarify that we have neither a proof of this claim nor a formalization of what "natural" or "practical" would mean.

Proceeding, Figure 7 shows three PSI-related functionalities that have arisen in the literature. Below each are inverters for it, first for honest party 2 and then for honest party 1, demonstrating invertibility of that functionality for both parties. The set U is the universe. We now discuss these in turn.

PSI. The PSI functionality $\mathsf{F}^{\mathrm{psi}}_U\colon \mathcal{P}(U) \times \mathcal{P}(U) \to (\mathcal{P}(U) \times \mathbb{N}) \times \mathbb{N}$ in the first panel is the same as in Figure 3, repeated for clarity. Here $S_1, S_2 \subseteq U$.

The inverter for party 2 takes as input an input set S_1 for party 1 and an output (I, s_2) for party 1, where I is the intersection of S_1 with some (unknown to the inverter) set S_2 of party 2, and $s_2 = |S_2|$. The inverter aims to construct some (any) set S'_2 of size s_2 such that $S_1 \cap S'_2 = I$. It does this as shown.

The inverter for party 1 is easier. It gets an input set S_2 for party 2 and an output s_1 that is the size of some (unknown to the inverter) set S_1 of party 1. It aims to construct some (any) set S_1' of size s_1, done as shown.

Both inverters are linear time. They are thus efficient as required for invertibility.

THRESHOLD PSI. The threshold-PSI (tPSI) functionality [26] $\mathsf{F}^{\mathrm{tpsi}}_{U,t} \colon \mathcal{P}(U) \times \mathcal{P}(U) \to ((\mathcal{P}(U) \cup \{\bot\}) \times \mathbb{N}) \times \mathbb{N}$ is parameterized, in addition to U, by an integer $t \geq 0$ which specifies the threshold that the cardinality of intersection of S_1 and S_2 must reach for the intersection to appear in the output of party 1. Clearly when $t = 0$, the tPSI functionality is same as the basic PSI functionality.

The inverter for party 2 takes input an input set S_1 for party 1 and an output (I, s_2) for party 1, where I is either the intersection of S_1 with some (unknown to the inverter) set S_2 of party 2 or is $\bot$, and $s_2 = |S_2|$. In the case that $I = \bot$, the inverter picks and returns some (any) set S_2' of size s_2. If $I \neq \bot$, it runs the PSI inverter. The inverter for party 1 is the same as for PSI. The inverters are again linear time.

CARDINALITY PSI. The cardinality-PSI (cPSI) functionality [26] $\mathsf{F}^{\mathrm{cpsi}}_{U} \colon \mathcal{P}(U) \times \mathcal{P}(U) \to (\mathbb{N} \times \mathbb{N}) \times \mathbb{N}$ provides the cardinality of the intersection, rather than the intersection itself, in the output for party 1.

The inverter for party 2 takes input an input set S_1 for party 1 and an output (n, s_2) for party 1, where n is the size of the intersection of S_1 with some (unknown to the inverter) set S_2 of party 2, and $s_2 = |S_2|$. The inverter aims to construct some (any) set S_2' of size s_2 such that $|S_1 \cap S_2'| = n$. It does this as shown. The inverter for party 1 is the same as for PSI, and as before the inverters are linear time.

3.4 General composition result

In practice, a 2PC protocol will be executed many times on different inputs. We want to prove that this is secure. To that end, we consider general composition and ask whether security for a single execution security implies security for multiple executions. As one might expect, a simple hybrid argument does work and the claim below formalizes just that. The proof is in Appendix F.

Theorem 3.5 *Let F be a functionality. Let $h \in \{1, 2\}$ be the honest party. Let Π be a protocol for F. Let A_{ini} be an adversary playing game $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F},\Pi,h}$. Then we can construct an adversary B_{ini}, also playing game $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F},\Pi,h}$ but making at most one* RUN *query, such that*

$$\mathbf{Adv}^{\mathrm{ini}}_{\mathsf{F},\Pi,h}(A_{\mathrm{ini}}) \leq \mathrm{Q}^{\mathrm{RUN}}(A_{\mathrm{ini}}) \cdot \mathbf{Adv}^{\mathrm{ini}}_{\mathsf{F},\Pi,h}(B_{\mathrm{ini}}) \; . \tag{6}$$

Additionally $\mathrm{Q}^{\mathrm{RO}}(B_{\mathrm{ini}}) = \mathrm{Q}^{\mathrm{RO}}(A_{\mathrm{ini}})$ and the running time of B_{ini} is about that of A_{ini}.

Asymptotically, this would end the question, but concretely it is more of a starting point, for it raises the question of showing security for multiple executions tightly, meaning with the same bound as for a single execution rather than with the linear degradation of the hybrid argument. In the following sections we will do this for OPRF and PSI protocols.

4 PSI from OPRFs

In this section, we evaluate the concrete security of the canonical OPRF-based PSI protocol of Hazay and Lindell [32]. To do this, we first give definitions for OPRFs. Then we prove InI security of the PSI protocol, based on the security of the underlying OPRF, with a reduction that is *tight*.

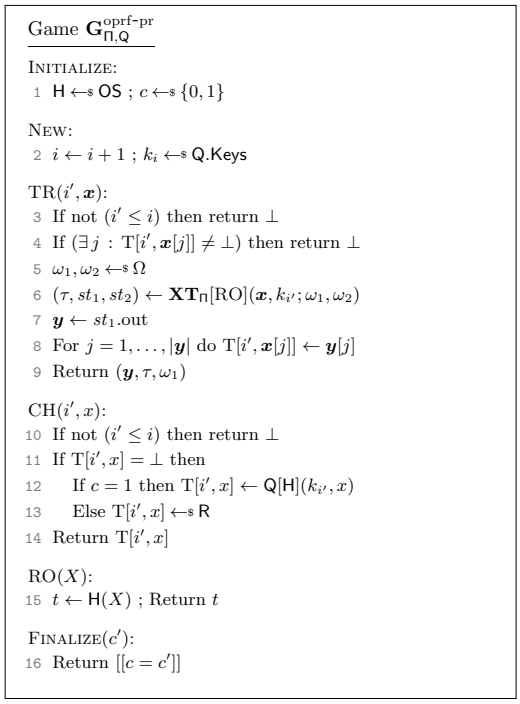


Figure 8: OPRF-PR game for pseudo-randomness (server side security) of an OPRF protocol Π. Here Q is the underlying PRF.

OBLIVIOUS PSEUDORANDOM FUNCTIONS. Let $\mathsf{Q}\colon [\mathsf{OS}] \times \mathsf{Keys} \times \mathsf{D} \to \mathsf{R}$ be a family of functions. The OPRF functionality $\mathsf{F}^{\text{oprf}}_{\mathsf{Q}}\colon [\mathsf{OS}] \times \mathsf{D}^* \times \mathsf{Keys} \to \mathsf{R}^* \times \mathbb{N}$ associated to Q was defined in Figure 3. We say that protocol Π is an OPRF for Q if it computes the functionality $\mathsf{F}^{\text{oprf}}_{\mathsf{Q}}$ with perfect correctness. (The latter condition can be relaxed but is met by the OPRFs we consider, so we require it for simplicity.) We say Π is an OPRF if it is an OPRF for some Q.

In an OPRF protocol, the server input is a secret key $k \in \mathsf{Keys}$. Conventionally, the client input would be a point in D, but we generalize this; in our setting the client input is a vector $\boldsymbol{x}$ over D. The client output is the vector $(\mathsf{Q}(k, \boldsymbol{x}[1]), \ldots, \mathsf{Q}(k, \boldsymbol{x}[|\boldsymbol{x}|]))$ and the server output is $|\boldsymbol{x}|$.

OPRF SECURITY. Let protocol Π be an OPRF for $\mathsf{Q}\colon [\mathsf{OS}] \times \mathsf{Keys} \times \mathsf{D} \to \mathsf{R}$. We separately define OPRF-security of Π for party 1 (client) and party 2 (server).

The client-security definition is simple, namely just InI-security as defined in Section 3. This says that the server cannot obtain information about the client input. This shows how we can leverage our definitional framework for OPRFs.

For server-security, we give a very simple definition of pseudorandomness that we call OPRF-PR. Namely, we take the game defining PRF security of function family Q in Figure 3 and simply add an oracle that allows the adversary to obtain transcripts of the protocol execution. The resulting game $\mathbf{G}^{\text{oprf-pr}}_{\Pi,\mathsf{Q}}$ is shown in Figure 8. Challenge oracle CH is as in Figure 3. The transcript oracle TR takes a vector $\boldsymbol{x}$ of client inputs and (line 6) executes the protocol to obtain a conversation transcript and final states of the parties. From the final states, it extracts the party outputs, and uses the client outputs to update the table that stores the challenge function. Note that these entries are always the real ones as computed by the protocol, meaning, if $c = 0$, the challenge entries are random but the ones created by the transcript oracle are still real. The advantage of adversary A_{oprf} is $\mathbf{Adv}^{\text{oprf-pr}}_{\Pi,\mathsf{Q}}(A_{\text{oprf}}) = 2\Pr[\mathbf{G}^{\text{oprf-pr}}_{\Pi,\mathsf{Q}}(A_{\text{oprf}})] - 1$.

PSI Protocol Π^{psi}

Based on: Function family $\mathsf{Q}\colon [\mathsf{OS}] \times \mathsf{Keys} \times U \to \mathsf{R}$ and OPRF Π^{oprf} for Q
Party 1 input: Set $S_1 \subseteq U$ of size s_1
Party 2 input: Set $S_2 \subseteq U$ of size s_2
Oracle: $\mathsf{H} \in \mathsf{OS}$ available to all parties.

1. Party 1 constructs its input vector $\boldsymbol{x} \leftarrow \mathsf{S2V}(S_1)$ for the OPRF protocol from the elements of its set S_1.
2. Party 2 picks a key $k \leftarrow\!\!\$\ \mathsf{Keys}$ for the PRF Q.
3. The parties run the OPRF Π^{oprf} with the input of party 1 being $\boldsymbol{x}$ and that of party 2 being k. The output of party 1 is $\boldsymbol{y} \in \mathsf{R}^{|\boldsymbol{x}|}$, and that of party 2 is $|\boldsymbol{x}| = s_1$.
4. Party 2 lets $Z = \{\, \mathsf{Q}[\mathsf{H}](k, s) : s \in S_2 \,\}$ and $\boldsymbol{z} \leftarrow \mathsf{S2V}(Z)$. It sends $\boldsymbol{z}$ to party 1.
5. Party 1 constructs intersection set I as $I \leftarrow \{\, \boldsymbol{x}[i] : 1 \le i \le s_1 \text{ and } \boldsymbol{y}[i] \in \mathsf{V2S}(\boldsymbol{z}) \,\}$. It also computes $|\boldsymbol{z}| = s_2$.

Party 1 output: Set $I \subseteq U$ and size s_2 of S_2
Party 2 output: Size s_1 of S_1.

Figure 9: PSI protocol Π^{psi} associated to function family Q and OPRF Π^{oprf} for Q.

A definition of pseudorandomness for OPRFs is also given in [55], but it is simulation-based and thus double-quantifier. Our simpler definition is single-quantifier.

The following says that if Π is a OPRF-PR-secure OPRF for a function family Q, then the latter is PRF-secure. The proof is trivial and is omitted.

Proposition 4.1 *Let protocol Π be an OPRF for function family $\mathsf{Q}\colon [\mathsf{OS}] \times \mathsf{Keys} \times \mathsf{D} \to \mathsf{R}$. Let A_{prf} be an adversary playing game $\mathbf{G}^{\mathrm{prf}}_{\mathsf{Q}}$. Then we can construct an adversary A_{oprf} playing game $\mathbf{G}^{\mathrm{oprf\text{-}pr}}_{\Pi,\mathsf{Q}}$ such that*

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{Q}}(A_{\mathrm{prf}}) \le \mathbf{Adv}^{\mathrm{oprf\text{-}pr}}_{\Pi,\mathsf{Q}}(A_{\mathrm{oprf}}) \ . \tag{7}$$

Adversary A_{oprf} has the same query profile and running time as A_{prf}, in particular making no TR *queries.*

PSI FROM OPRF. Now that we have security definitions for OPRFs, we analyze the InI security of the classic OPRF-based protocol from [32]. The protocol, which we denote Π^{psi}, is shown in Figure 9. It is associated to a family of functions $\mathsf{Q}\colon [\mathsf{OS}] \times \mathsf{Keys} \times U \to \mathsf{R}$ and an OPRF Π^{oprf} for Q. The random oracle $\mathsf{H} \in \mathsf{OS}$ is used by Π^{oprf} and Q, both of which are used by Π^{psi}. The universe U is the domain of Q. The protocol Π^{psi} computes the functionality $\mathsf{F}^{\mathrm{psi}}_U$ defined in Figure 3. The following says that OPRF tightly implies PSI, meaning there is a tight reduction from Π^{oprf} to Π^{psi}. Note that PRF security of Q, as required by part 1, is not an extra assumption due to Proposition 4.1. The proof is in Appendix G.

Theorem 4.2 *Let $U \subseteq \{0,1\}^*$ be a set (the universe), and $\mathsf{F} = \mathsf{F}^{\mathrm{psi}}_U$ the associated PSI functionality. Let Π^{oprf} be an OPRF for a family of functions $\mathsf{Q}\colon [\mathsf{OS}] \times \mathsf{Keys} \times U \to \mathsf{R}$. Let $\Pi = \Pi^{\mathrm{psi}}$ be the PSI protocol built from Q and Π^{oprf} as in Figure 9. Then:*

1. *Π is correct for F if Q is a PRF: Let A_{psi} be an adversary playing game $\mathbf{G}^{\mathrm{corr}}_{\mathsf{F},\Pi}$. Then we can construct an adversary A_{prf} such that*

$$\mathbf{Adv}^{\mathrm{corr}}_{\mathsf{F},\Pi}(A_{\mathrm{psi}}) \le \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{Q}}(A_{\mathrm{prf}}) + \sum_{i=1}^{q} \frac{s_{i,1}s_{i,2}}{|\mathsf{R}|} \ . \tag{8}$$

Here $q = \mathrm{Q}^{\mathrm{Run}}(A_{\mathrm{psi}})$ and $s_{i,j}$ is the upper bound on the size of party j in the i-th Run *query. Also $\mathrm{Q}^{\mathrm{New}}(A_{\mathrm{prf}}) = q$ and $\mathrm{Q}^{\mathrm{CH}}(A_{\mathrm{prf}}) = \sum_{i=1}^{q}(s_{i,1} + s_{i,2})$ and $\mathrm{Q}^{\mathrm{RO}}(A_{\mathrm{prf}}) = \mathrm{Q}^{\mathrm{RO}}(A_{\mathrm{psi}})$. The running time of A_{prf} is about that of A_{psi}.*

2. $\mathsf{\Pi}$ *provides InI client security if* $\mathsf{\Pi}^{\text{oprf}}$ *does: Let* A_{psi} *be an adversary playing game* $\mathbf{G}^{\text{ini}}_{\mathsf{F},\mathsf{\Pi},1}$. *Then we can construct an adversary* A_{oprf} *playing game* $\mathbf{G}^{\text{ini}}_{\mathsf{F}^{\text{oprf}}_{\mathsf{Q}},\mathsf{\Pi}^{\text{oprf}},1}$ *such that*

$$\mathbf{Adv}^{\text{ini}}_{\mathsf{F},\mathsf{\Pi},1}(A_{\text{psi}}) \le \mathbf{Adv}^{\text{ini}}_{\mathsf{F}^{\text{oprf}}_{\mathsf{Q}},\mathsf{\Pi}^{\text{oprf}},1}(A_{\text{oprf}}) \ . \tag{9}$$

Adversary A_{oprf} *makes the same number of* RUN *queries as* A_{psi}*, with the vector in each of length the (common) size of the two client sets in the corresponding query of* A_{psi}*. Also,* $\mathrm{Q}^{\text{RO}}(A_{\text{oprf}}) \le \mathrm{Q}^{\text{RO}}(A_{\text{psi}}) + \sum_{i=1}^{q} s_{i,2}$ *where* $q = \mathrm{Q}^{\text{RUN}}(A_{\text{psi}})$ *and* $s_{i,2}$ *is the upper bound on the size of party* 2*'s set in the* i*-th* RUN *query. The running time of* A_{oprf} *is about that of* A_{psi}.

3. $\mathsf{\Pi}$ *provides InI server security if* $\mathsf{\Pi}^{\text{oprf}}$ *is OPRF-PR secure: Let* A_{psi} *be an adversary playing game* $\mathbf{G}^{\text{ini}}_{\mathsf{F},\mathsf{\Pi},2}$. *Then we can construct an adversary* A_{oprf} *playing game* $\mathbf{G}^{\text{oprf-pr}}_{\mathsf{\Pi}^{\text{oprf}},\mathsf{Q}}$ *such that*

$$\mathbf{Adv}^{\text{ini}}_{\mathsf{F},\mathsf{\Pi},2}(A_{\text{psi}}) \le 2 \cdot \mathbf{Adv}^{\text{oprf-pr}}_{\mathsf{\Pi}^{\text{oprf}},\mathsf{Q}}(A_{\text{oprf}}) \ . \tag{10}$$

Let $q = \mathrm{Q}^{\text{RUN}}(A_{\text{psi}})$*. Then* $\mathrm{Q}^{\text{NEW}}(A_{\text{oprf}}) = \mathrm{Q}^{\text{TR}}(A_{\text{oprf}}) = q$ *and* $\mathrm{Q}^{\text{CH}}(A_{\text{oprf}}) \le \sum_{i=1}^{q} s_{i,2}$ *and* $\mathrm{Q}^{\text{RO}}(A_{\text{oprf}}) = \mathrm{Q}^{\text{RO}}(A_{\text{psi}})$ *where* $s_{i,2}$ *is an upper bound on the size of party* 2*'s set(s) in the* i*-th* RUN *query of* A_{psi}*. The running time of* A_{oprf} *is about that of* A_{psi}.

The above tightly bounds the InI security of $\mathsf{\Pi}^{\text{psi}}$ via the OPRF security of $\mathsf{\Pi}^{\text{oprf}}$. So if we can concretely bound the OPRF security of $\mathsf{\Pi}^{\text{oprf}}$, we can pick parameters to use in practice for $\mathsf{\Pi}^{\text{psi}}$ to guarantee a desired level of security. Accordingly we now turn to proving security with concrete bounds for a canonical OPRF.

5 Computational problems over the group

We will show concrete OPRF-PR-security of the 2H-DH OPRF based on the hardness of a variety of different computational problems over the underlying group. In each case, we will give explicit bounds on the OPRF-PR-advantage as a function of the advantage in solving the group problem. These bounds will differ, and the intent is exactly to showcase how the choice of group problem affects the bound. In this section we define the relevant computational problems and give relations between them.

THE PROBLEMS. Let $\mathbb{G} = \langle g \rangle$ be a group of prime order p with generator g. The problems we consider are CDH, DDH, V-CDH, which are in the single-user setting, multi-user versions CDH-MU, V-CDH-MU, DDH-MU, and multi-user with corruptions versions CDH-MUC and V-CDH-MUC. All problems are defined via the games in Figure 10. Throughout Figure 10, writing game names next to an oracle means that only the named games include the oracle. If there is no annotation, all the games include that oracle. For $\text{xx} \in \{\text{cdh}, \text{v-cdh}, \text{cdh-mu}, \text{v-cdh-mu}, \text{cdh-muc}, \text{v-cdh-muc}\}$ we define the advantage of an adversary A_{xx} by $\mathbf{Adv}^{\text{xx}}_{\mathbb{G},g,p}(A_{\text{xx}}) = \Pr[\mathbf{G}^{\text{xx}}_{\mathbb{G},g,p}(A_{\text{xx}})]$. For $\text{xx} \in \{\text{ddh}, \text{ddh-mu}\}$ we define the advantage of an adversary A_{xx} by $\mathbf{Adv}^{\text{xx}}_{\mathbb{G},g,p}(A_{\text{xx}}) = 2\Pr[\mathbf{G}^{\text{xx}}_{\mathbb{G},g,p}(A_{\text{xx}})] - 1$.

Now let us explain. CDH, DDH are the standard computational and decisional Diffie-Hellman problems. V-CDH is the verifiable computational Diffie-Hellman problem. It asks to solve CDH for group elements $K = g^k$ and B when given access to an oracle $\text{DDHO}(Z')$ which can check if $Z' = B^k$. It is similar to the commonly used *strong* CDH problem from [1]. Compared to that problem, the DDHO oracle only allows to check the CDH solution (hence we call it verifiable). This makes the assumption weaker than that of [1] which in turn is weaker than Gap-CDH [48]. But it is sufficient for our results.

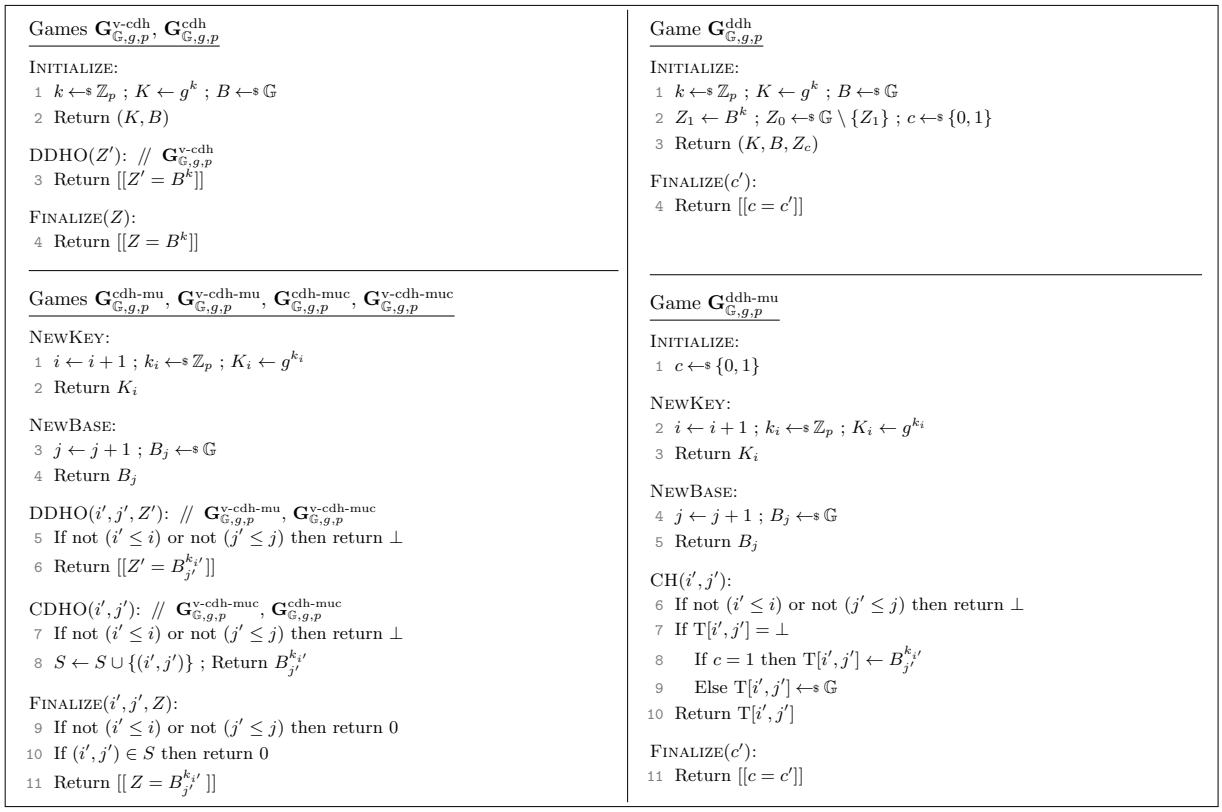


Figure 10: Here $\mathbb{G}$ is a group with prime order p and generator g. **Left:** Games for the CDH and V-CDH problems (top) and CDH-MU, V-CDH-MU, CDH-MUC and V-CDH-MUC problems (bottom). The DDHO oracle is only present in games $\mathbf{G}^{\text{v-cdh}}_{\mathbb{G},g,p}$, $\mathbf{G}^{\text{v-cdh-mu}}_{\mathbb{G},g,p}$ and $\mathbf{G}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}$. The CDHO oracle is only present in games $\mathbf{G}^{\text{cdh-muc}}_{\mathbb{G},g,p}$ and $\mathbf{G}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}$. **Right:** Game for the DDH problem (top) and the DDH-MU problem (bottom).

CDH, DDH, V-CDH are in the single-user setting. CDH-MU, DDH-MU, V-CDH-MU are multi-user extensions of them, and CDH-MUC, V-CDH-MUC extend CDH-MU, V-CDH-MU, respectively, to allow corruptions. The game for DDH-MU is given on the right part of Figure 10. It is defined as an interactive game which will be useful for our proofs. It samples a random bit c and then provides the adversary access to oracles NewKey, NewBase and CH. (The names reflect their usage in our security proofs, i.e., group elements output by NewKey can be viewed as OPRF keys and those output by NewBase will be used as random oracle outputs.) The i^{th} query to NewKey and j^{th} query to NewBase returns a random group element $K_i = g^{k_i}$ and B_j, respectively. For a pair of indices (i', j'), the CH oracle outputs either $B_{j'}^{k_{i'}}$ or a random group element, where the same challenge bit c is used across all queries. The table T records these queries to avoid trivial wins via repeated queries.

The other problems are given on the left part of Figure 10. Starting with the CDH multi-user problem (CDH-MU), we can construct games for other problems by giving additional power (through oracles or lesser restrictions) to the adversaries and thereby strengthening the assumption (or making the problem easier). The NewKey and NewBase oracles are as above. The CDH-MU problem asks to find $B_{j'}^{k_{i'}}$ for any pair (i', j'). The game for the V-CDH multi-user problem (V-CDH-MU) additionally allows access to an oracle DDHO(i', j', Z) which checks if $(Z = B_{j'}^{k_{i'}})$. For problems CDH-MUC and V-CDH-MUC, where C stands for "Corruption", the respective game additionally gives the adversary an oracle CDHO which can be queried for a pair of indices (i', j') and returns the CDH solution for that pair. These problems are similar (but

Problem P	Oracles available for adversaries	
	DDHO	CDHO
CDH-MU	×	×
V-CDH-MU	✓	×
CDH-MUC	×	✓
V-CDH-MUC	✓	✓

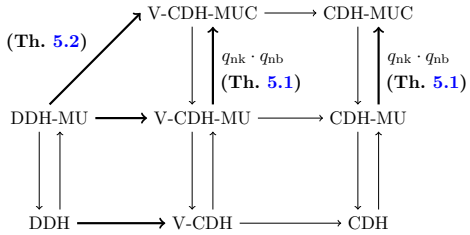


Figure 11: **Top:** Summary of oracles available to adversaries when playing the game $\mathbf{G}^{\mathrm{P}}_{\mathbb{G}}$. **Bottom:** Diagram showing relations between the assumptions. The arrows represent implications. Here $q_{\text{nk}} = \mathrm{Q}^{\textsc{NewKey}}(A)$ and $q_{\text{nb}} = \mathrm{Q}^{\textsc{NewBase}}(A)$ where A is playing the game $\mathbf{G}^{\mathrm{P}}_{\mathbb{G},g,p}$ for $\mathrm{P} \in \{\text{V-CDH-MUC}, \text{CDH-MUC}\}$.

weaker) than the One-More CDH problem that has appeared in the context of OPRFs and PSI. We prevent trivial wins using the set S that records CDHO queries.

The table at the top of Figure 11 is intended to clarify the problems by showing which oracles are available to the adversary in which case.

Relations between problems. We will prove security of the 2H-DH OPRF directly under some assumptions and get bounds under others via relations between the assumptions. Figure 11 shows a diagram of the relations. Here "$\mathrm{P}_1 \to \mathrm{P}_2$" is an implication, and means that, if P_1 is hard in group $\mathbb{G}$ then P_2 is also hard in $\mathbb{G}$. If an arrow is annotated with a value, for example $q_{\text{nk}} \cdot q_{\text{nb}}$, it means the reduction looses this factor. If there is no annotation, the reduction is tight. Some of the implications are trivial and easy to see, for example, V-CDH-MUC $\to$ CDH-MUC and CDH-MUC $\to$ CDH-MU. Standard re-randomization allows us to tightly obtain DDH $\to$ DDH-MU, CDH $\to$ CDH-MU and V-CDH $\to$ V-CDH-MU. The reductions V-CDH $\to$ V-CDH-MUC and CDH $\to$ CDH-MUC as well as DDH $\to$ V-CDH-MUC are more interesting. The first two are captured by the following theorem, whose proof is in Appendix H.

Theorem 5.1 *Let $\mathbb{G} = \langle g \rangle$ be a group with prime order p. Let A_{v} and A be adversaries playing the $\mathbf{G}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}$ game and the $\mathbf{G}^{\text{cdh-muc}}_{\mathbb{G},g,p}$ game, respectively. Then we can construct adversaries $A_{\text{v-cdh}}$ and A_{cdh} playing the $\mathbf{G}^{\text{v-cdh}}_{\mathbb{G},g,p}$ and $\mathbf{G}^{\text{cdh}}_{\mathbb{G},g,p}$ games, respectively, such that*

$$\mathbf{Adv}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}(A_{\mathrm{v}}) \leq \mathrm{Q}^{\textsc{NewKey}}(A_{\mathrm{v}}) \cdot \mathrm{Q}^{\textsc{NewBase}}(A_{\mathrm{v}}) \cdot \mathbf{Adv}^{\text{v-cdh}}_{\mathbb{G},g,p}(A_{\text{v-cdh}}) \ , \tag{11}$$

$$\mathbf{Adv}^{\text{cdh-muc}}_{\mathbb{G},g,p}(A) \leq \mathrm{Q}^{\textsc{NewKey}}(A) \cdot \mathrm{Q}^{\textsc{NewBase}}(A) \cdot \mathbf{Adv}^{\text{cdh}}_{\mathbb{G},g,p}(A_{\text{cdh}}) \ . \tag{12}$$

The running time of $A_{\text{v-cdh}}$ is about that of A_{v} plus the time for $(\mathrm{Q}^{\textsc{NewKey}}(A_{\mathrm{v}}) + \mathrm{Q}^{\textsc{NewBase}}(A_{\mathrm{v}}) + \mathrm{Q}^{\mathrm{CDHO}}(A_{\mathrm{v}}) + \mathrm{Q}^{\mathrm{DDHO}}(A_{\mathrm{v}}))$ group exponentiations, and the running time of A_{cdh} is about that of A plus the time for $(\mathrm{Q}^{\textsc{NewKey}}(A) + \mathrm{Q}^{\textsc{NewBase}}(A) + \mathrm{Q}^{\mathrm{CDHO}}(A))$ group exponentiations.

Curiously, we can show DDH $\to$ V-CDH-MUC with a tight reduction. This is captured by the following, whose proof is in Appendix I.

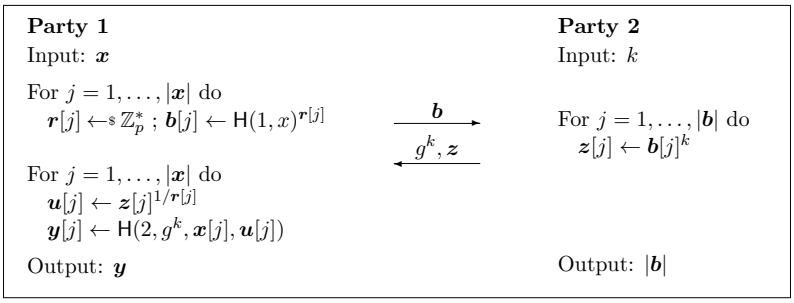


Figure 12: 2H-DH OPRF protocol Π^{2HDH}.

Theorem 5.2 *Let $\mathbb{G} = \langle g \rangle$ be a group with prime order p. Let A_{v} be an adversary playing the $\mathbf{G}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}$ game. Then we can construct an adversary A_{ddh} playing the $\mathbf{G}^{\mathrm{ddh}}_{\mathbb{G},g,p}$ game such that*

$$\mathbf{Adv}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}(A_{\mathrm{v}}) \leq \mathbf{Adv}^{\mathrm{ddh}}_{\mathbb{G},g,p}(A_{\mathrm{ddh}}) + \frac{\mathrm{Q}^{\mathrm{DDHO}}(A_{\mathrm{v}}) + 1}{p} \,. \tag{13}$$

The running time of A_{ddh} is about that of A_{v} plus the time for at most $(\mathrm{Q}^{\mathrm{NewKey}}(A_{\mathrm{v}}) + 2 \cdot \mathrm{Q}^{\mathrm{NewBase}}(A_{\mathrm{v}}) + 2 \cdot \mathrm{Q}^{\mathrm{CDHO}}(A_{\mathrm{v}}) + 2 \cdot \mathrm{Q}^{\mathrm{DDHO}}(A_{\mathrm{v}}))$ group exponentiations.

6 Security of 2H-DH OPRF

We have shown (Theorem 4.2) that PSI can be built *tightly* from an OPRF. Now we turn to seeing how tightly we can build OPRFs based on algebraic assumptions. For this purpose we consider 2H-DH [36], a leading and very efficient OPRF. We will showcase how its security can be proven under different algebraic assumptions with different degrees of tightness. We note that the 2H-DH OPRF has many applications beyond PSI [22, 36, 37, 23], making our results about it of independent interest.

2H-DH OPRF. We fix a group $\mathbb{G} = \langle g \rangle$ of prime order p with generator g. We also fix an integer $\ell \geq 1$. Recall that we associated to $\mathbb{G}, \ell$ the family of functions $\mathsf{2HDH}\colon [\mathsf{OS}] \times \mathbb{Z}_p \times \{0,1\}^* \to \{0,1\}^\ell$ shown in Figure 3. It uses a random oracle $\mathsf{H} \in \mathsf{OS}$ which specifies two sub-functions: $\mathsf{H}(1,\cdot)\colon \{0,1\}^* \to \mathbb{G}$ and $\mathsf{H}(2,\cdot,\cdot,\cdot)\colon \mathbb{G} \times \mathbb{G} \times \{0,1\}^* \times \mathbb{G} \to \{0,1\}^\ell$. Succinctly, $\mathsf{2HDH}[\mathrm{RO}](k,x) = \mathsf{H}(2, g^k, x, \mathsf{H}(1,x)^k)$. The 2H-DH protocol is shown in Figure 12. It realizes the functionality $\mathsf{F}^{\mathrm{oprf}}_{\mathsf{2HDH}}$ with perfect correctness. The client (party 1) has input a vector $\boldsymbol{x}$ over $\{0,1\}^*$, and the server has input a key $k \in \mathbb{Z}_p$ for $\mathsf{2HDH}$. The vector $\boldsymbol{r}$ holds the blinding factors.

InI security for the client. We first want to show InI security of the Π^{2HDH} OPRF protocol for an honest client (party 1). In our concrete setting, we aim to give a concrete bound on the ini-advantage of adversary A_{ini}. This turns out to need a bit of care. The first thought may be that the advantage is unconditionally zero, regardless of the running time or number of oracle queries of the adversary, because the randomness of the blinding factors means that the entries of the vector $\boldsymbol{b}$ are random and independent group elements. However, if $\mathrm{RO}(1,x) = 1$ is the identity element of the group, then $b = \mathrm{RO}(1,x)^r = 1$ is also the identity for all $r \in \mathbb{Z}_p^*$, and is not uniformly distributed. Otherwise, $\mathrm{RO}(1,x)$ is a generator, and b is uniformly distributed. (But over $\mathbb{G}^*$, not $\mathbb{G}$.) The advantage thus depends on whether or not there is, in the execution of A_{ini} with the game, some query x to $\mathrm{RO}(1,\cdot)$ that returns 1. The probability of this depends on the number of RO queries made, either directly by A_{ini} or by the protocol in the execution of the game with A_{ini}. Recall that by our conventions, $\mathrm{Q}^{\mathrm{RO}}(A_{\mathrm{ini}})$ counts both. This term now enters the bound, which

Problem P	$\mathbf{B}(\epsilon', \{q_{\rm ro}, q_{\rm n}, q_{\rm tr}\})$	Number of queries			
		NewKey	NewBase	DDHO	CDHO
CDH	$2 \cdot (q_{\rm ro}^2 q_{\rm n} \cdot \epsilon' + \alpha_c)$	–	–	–	–
V-CDH	$2 \cdot (q_{\rm ro} q_{\rm n} \cdot \epsilon' + \alpha_c)$	–	–	$q_{\rm ro}$	–
CDH-MUC	$2 \cdot (q_{\rm ro} \cdot \epsilon' + \alpha_c)$	$q_{\rm n}$	$q_{\rm ro}$	–	$q_{\rm tr} \cdot \ell_{\rm tr}$
V-CDH-MUC	$2 \cdot (\epsilon' + \alpha_c)$	$q_{\rm n}$	$q_{\rm ro}$	$q_{\rm ro}$	$q_{\rm tr} \cdot \ell_{\rm tr}$
DDH	$2 \cdot (\epsilon' + \alpha_d)$	–	–	–	–

Figure 13: **Our results showing OPRF-PR security of the 2H-DH OPRF.** For different choices of the assumed-hard problem P, we show the bound $\mathbf{B}(\epsilon', \{q_{\rm ro}, q_{\rm n}, q_{\rm tr}\})$ on the advantage of an OPRF-PR adversary A as a function of the advantage $\epsilon' = \mathbf{Adv}^{\rm P}_{\mathbb{G}}(A')$ of the constructed adversary A' in solving problem P in group $\mathbb{G}$. We also show the query profile of A'. Here $q_{\rm ro} = \mathrm{Q}^{\rm RO}(A)$, $q_{\rm n} = \mathrm{Q}^{\rm New}(A)$, $q_{\rm tr} = \mathrm{Q}^{\rm TR}(A)$, $\alpha_c = (q_{\rm ro} \cdot q_{\rm rn})/p$, $\alpha_d = (q_{\rm ro} \cdot q_{\rm rn} + q_{\rm ro} + 1)/p$ and $\ell_{\rm tr}$ is the maximum length of vectors queried to TR. Adversaries A, A' have about the same running time. A dash (–) means that the oracle is not present for P.

in particular means security is not unconditional after all. (To guarantee a low advantage via the Theorem, one must assume $\mathrm{Q}^{\rm RO}(A_{\rm ini})$ is sufficiently less than p.) The proof of the following is in Appendix J.

Theorem 6.1 *Let $\mathbb{G} = \langle g \rangle$ be a group with prime order p, and $\ell \geq 1$ an integer. Let* 2HDH *be the associated 2H-DH family of functions as per Figure 3. Let $\Pi^{\rm 2HDH}$ be the associated 2H-DH OPRF protocol as per Figure 12 and let $\mathsf{F} = \mathsf{F}^{\rm oprf}_{\rm 2HDH}$ be the OPRF functionality that $\Pi^{\rm 2HDH}$ computes. Let $A_{\rm ini}$ be an adversary playing game $\mathbf{G}^{\rm ini}_{\mathsf{F},\Pi^{\rm 2HDH},1}$. Then*

$$\mathbf{Adv}^{\rm ini}_{\mathsf{F},\Pi^{\rm 2HDH},1}(A_{\rm ini}) \leq \frac{\mathrm{Q}^{\rm RO}(A_{\rm ini})}{p} \ . \tag{14}$$

We note that if we set the range of $\mathsf{H}(1,\cdot)$ to $\mathbb{G}^*$ rather than $\mathbb{G}$ then the above advantage would be always zero. However, we would then incur other terms in security bounds, so we have stayed with the more conventional choice.

OPRF-PR security for the server. We bound the adversary advantage via the advantage to solve the different problems from Section 5 on the underlying group $\mathbb{G} = \langle g \rangle$, showcasing how the bounds change across these problems. The proof of the following is in Appendix K. Figure 13 summarizes the bounds, and the resource usage of the constructed adversaries, and is a more precise version of column 2 of Figure 2.

Theorem 6.2 *Let $\mathbb{G} = \langle g \rangle$ be a group with prime order p, and $\ell \geq 1$ an integer. Let* 2HDH *be the associated 2H-DH family of functions as per Figure 3. Let $\Pi^{\rm 2HDH}$ be the associated 2H-DH OPRF protocol as per Figure 12. Let $A_{\rm oprf}$ be an adversary playing game $\mathbf{G}^{\rm oprf\text{-}pr}_{\Pi^{\rm 2HDH},\rm 2HDH}$, and let* $\mathrm{xx} \in \{\mathrm{ddh}, \mathrm{cdh}, \text{v-cdh}, \text{cdh-muc}, \text{v-cdh-muc}\}$*. Then we can construct an adversary $A_{\rm xx}$ playing game $\mathbf{G}^{\rm xx}_{\mathbb{G},g,p}$ such that*

$$\begin{aligned} \mathbf{Adv}^{\rm oprf\text{-}pr}_{\Pi^{\rm 2HDH},\rm 2HDH}(A_{\rm oprf}) &\leq 2 \cdot \boldsymbol{\mu}^{\rm xx}(q_{\rm ro}, q_{\rm n}) \cdot \mathbf{Adv}^{\rm xx}_{\mathbb{G},g,p}(A_{\rm xx}) \\ &\quad + \frac{2 \cdot \boldsymbol{\delta}^{\rm xx}(q_{\rm ro}, q_{\rm n})}{p} \ , \end{aligned} \tag{15}$$

where $q_{\mathrm{n}} = \mathrm{Q}^{\mathrm{New}}(A_{\mathrm{oprf}})$ *and* $q_{\mathrm{ro}} = \mathrm{Q}^{\mathrm{RO}}(A_{\mathrm{oprf}})$. *Further*

$$\boldsymbol{\mu}^{\mathrm{xx}}(q_{\mathrm{ro}}, q_{\mathrm{n}}) = \begin{cases} q_{\mathrm{ro}}^2 q_{\mathrm{n}} & \text{if } \mathrm{xx} = \mathrm{cdh} \\ q_{\mathrm{ro}} q_{\mathrm{n}} & \text{if } \mathrm{xx} = \text{v-cdh} \\ q_{\mathrm{ro}} & \text{if } \mathrm{xx} = \text{cdh-muc} \\ 1 & \text{if } \mathrm{xx} \in \{\text{v-cdh-muc}, \mathrm{ddh}\} \end{cases} \tag{16}$$

and

$$\boldsymbol{\delta}^{\mathrm{xx}}(q_{\mathrm{ro}}, q_{\mathrm{n}}) = \begin{cases} q_{\mathrm{ro}} \cdot q_{\mathrm{n}} & \text{if } \mathrm{xx} \in \{\mathrm{cdh}, \text{v-cdh}, \text{cdh-muc}, \text{v-cdh-muc}\} \\ q_{\mathrm{ro}} \cdot q_{\mathrm{n}} + q_{\mathrm{ro}} + 1 & \text{if } \mathrm{xx} = \mathrm{ddh} \end{cases}$$

with resources

$$\begin{aligned} \mathrm{Q}^{\mathrm{NewKey}}(A_{\text{cdh-muc}}) = \mathrm{Q}^{\mathrm{NewKey}}(A_{\text{v-cdh-muc}}) &= \mathrm{Q}^{\mathrm{New}}(A_{\mathrm{oprf}}) \\ \mathrm{Q}^{\mathrm{NewBase}}(A_{\text{cdh-muc}}) = \mathrm{Q}^{\mathrm{NewBase}}(A_{\text{v-cdh-muc}}) &\le \mathrm{Q}^{\mathrm{RO}}(A_{\mathrm{oprf}}) \ , \\ \mathrm{Q}^{\mathrm{CDHO}}(A_{\text{cdh-muc}}) = \mathrm{Q}^{\mathrm{CDHO}}(A_{\text{v-cdh-muc}}) &\le \ell_{\mathrm{tr}} \cdot \mathrm{Q}^{\mathrm{TR}}(A_{\mathrm{oprf}}) \ , \\ \mathrm{Q}^{\mathrm{DDHO}}(A_{\text{v-cdh}}) = \mathrm{Q}^{\mathrm{DDHO}}(A_{\text{v-cdh-muc}}) &\le \mathrm{Q}^{\mathrm{RO}}(A_{\mathrm{oprf}}) \ , \end{aligned}$$

where ℓ_{tr} *is the maximum length of vectors queried to the* TR *oracle and the running times of all adversaries are about that of* A_{oprf}*, except that* A_{cdh} *and* $A_{\text{v-cdh}}$ *additionally perform at most* $\mathrm{Q}^{\mathrm{New}}(A_{\mathrm{oprf}}) + \mathrm{Q}^{\mathrm{RO}}(A_{\mathrm{oprf}}) + \ell_{\mathrm{tr}} \cdot \mathrm{Q}^{\mathrm{TR}}(A_{\mathrm{oprf}})$ *group exponentiations and* A_{ddh} *additionally performs at most* $\mathrm{Q}^{\mathrm{New}}(A_{\mathrm{oprf}}) + 2 \cdot \mathrm{Q}^{\mathrm{RO}}(A_{\mathrm{oprf}}) + 2 \cdot \ell_{\mathrm{tr}} \cdot \mathrm{Q}^{\mathrm{TR}}(A_{\mathrm{oprf}})$ *group exponentiations.*

Note that the factor $\mathrm{Q}^{\mathrm{NewBase}}(A)$ from the relations in Figure 11 translates to $\mathrm{Q}^{\mathrm{RO}}(A)$ in Theorem 6.2. The bound for DDH follows from combining Eq. (16) and Theorem 5.2.

With these results on the security of Π^{2HDH} and Theorem 4.2, we can show concrete bounds on the InI security of the DH-PSI protocol. (By the latter we mean the PSI protocol in Figure 9 when using Π^{2HDH} as the underlying OPRF.) In Figure 15 we depict our bounds, and the resource usage of the corresponding adversaries, for DH-PSI. This is a more precise and complete version of column 3 in Figure 2.

7 The Salted-DH PSI Protocol

We give a new PSI protocol. It has a proof with a *tight* reduction to the V-CDH (and hence also DDH) assumption and achieves better bounds for the CDH assumption than the previous protocol. Yet it has essentially the same computational cost as the OPRF-PSI when the OPRF is 2H-DH. The communication cost is more by just a constant (256 bits in practice) that does not depend on the sizes of the sets in the protocol.

The protocol is presented in Figure 14. We have fixed a group $\mathbb{G}$ of prime order p. The protocol is parameterized by a salt length sl and a hash-output length hl. (The latter only impacts correctness, not security.) Compared to 2H-DH based PSI, the increase in computation is just that the parties need to hash slightly longer strings, which has negligible cost relative to the cost of the group exponentiations, which is the same in both protocols. Communication increases by just sl, irrespective of the sizes of the sets involved.

The following Theorem establishes correctness and security of the protocol. The main claim is the third, showing security for the server based only on the V-CDH assumption. The added term is

Salted-DH PSI Protocol $\Pi^{\text{salt-psi}}$

Parameters: Salt length sl, hash length hl, group $\mathbb{G}$ of prime order p with generator g
Party 1 input: Set $S_1 \subseteq \{0,1\}^*$ of size s_1
Party 2 input: Set $S_2 \subseteq \{0,1\}^*$ of size s_2
Oracles: $\mathsf{H}_1 \colon \{0,1\}^{sl} \times \mathbb{G} \to \mathbb{G}$ and $\mathsf{H}_2 \colon \{0,1\}^{sl} \times \mathbb{G} \times \{0,1\}^* \times \mathbb{G} \to \{0,1\}^{hl}$

1. Party 1 casts the items in its set S_1 as a vector $\boldsymbol{x}_1 \leftarrow \mathsf{S2V}(S_1)$ of length s_1. It picks a salt $\eta \leftarrow\!\!\$\ \{0,1\}^{sl}$ and picks blinding exponents $r_1, \ldots, r_{s_1} \leftarrow\!\!\$\ \mathbb{Z}_p^*$. It then sets $\boldsymbol{a}_1[i] \leftarrow \mathsf{H}_1(\eta, \boldsymbol{x}_1[i])^{r_i}$ for $i = 1, \ldots, s_1$. It sends $(\eta, \boldsymbol{a}_1)$ to Party 2.
2. Party 2 picks a key $k \leftarrow\!\!\$\ \mathbb{Z}_p$ and computes $K \leftarrow g^k$. It lets $\boldsymbol{b}[i] \leftarrow \boldsymbol{a}_1[i]^k$ for $i = 1, \ldots, |\boldsymbol{a}_1|$. It randomly permutes the items in its set S_2 to get a vector $\boldsymbol{x}_2 \leftarrow\!\!\$\ \mathsf{S2V}(S_2)$ of length s_2. It lets $\boldsymbol{c}[i] \leftarrow \mathsf{H}_1(\eta, \boldsymbol{x}_2[i])^k$ and $\boldsymbol{a}_2[i] \leftarrow \mathsf{H}_2(\eta, K, \boldsymbol{x}_2[i], \boldsymbol{c}[i])$ for $i = 1, \ldots, s_2$. It sends $(K, \boldsymbol{b}, \boldsymbol{a}_2)$ to Party 1. It then halts with $s_1 = |\boldsymbol{a}_1|$ as its protocol output.
3. Party 1 constructs intersection set I via

   For all $i = 1, \ldots, s_1$ do
   $t_i \leftarrow r_i^{-1} \bmod p$; $\boldsymbol{d}[i] \leftarrow \boldsymbol{b}[i]^{t_i}$ // $\boldsymbol{d}[i] = \mathsf{H}_1(\eta, \boldsymbol{x}_1[i])^k$
   If $\mathsf{H}_2(\eta, K, \boldsymbol{x}_1[i], \boldsymbol{d}[i]) \in \mathsf{V2S}(\boldsymbol{a}_2)$ then $I \leftarrow I \cup \{\boldsymbol{x}_1[i]\}$.

   It outputs I as the intersection of S_1 and S_2, and halts.

Figure 14: Salted DH PSI protocol $\Pi^{\text{salt-psi}}$.

easily made negligible by picking a non-trivial salt length; in practice, $sl = 256$ will do. The result is that the reduction is tight. The proof is in Appendix L.

Theorem 7.1 *Let $\mathbb{G}$ be a group of prime order p with generator g. Let $hl, sl \geq 0$ be integers. Let $\mathsf{\Pi} = \Pi^{\text{salt-psi}}$ be the associated PSI protocol as per Figure 14. Let F be the PSI functionality over universe $\{0,1\}^*$. Below, M denotes an upper bound on the sum, across all* Run *queries of adversaries A_{corr} and A_{ini}, of the sizes of the sets in these queries.*

1. *Correctness: Let A_{corr} be an adversary playing game $\mathbf{G}^{\text{corr}}_{\mathsf{F},\mathsf{\Pi}}$. Then*

$$\mathbf{Adv}^{\text{corr}}_{\mathsf{F},\mathsf{\Pi}}(A_{\text{corr}}) \leq M^2/2^{hl} \ . \tag{17}$$

2. *Security for the client: Let A_{ini} be an adversarial server, meaning an adversary playing game $\mathbf{G}^{\text{ini}}_{\mathsf{F},\mathsf{\Pi},1}$. Then*

$$\mathbf{Adv}^{\text{ini}}_{\mathsf{F},\mathsf{\Pi},1}(A_{\text{ini}}) \leq \frac{\mathrm{Q}^{\text{RO}}(A_{\text{ini}})}{p} \ . \tag{18}$$

3. *Security for the server: Let A_{ini} be an adversarial client, meaning an adversary playing game $\mathbf{G}^{\text{ini}}_{\mathsf{F},\mathsf{\Pi},2}$. Then we can construct an adversary A_{xx} playing game $\mathbf{G}^{\text{xx}}_{\mathbb{G},g,p}$ such that*

$$\begin{aligned}\mathbf{Adv}^{\text{ini}}_{\mathsf{F},\mathsf{\Pi},2}(A_{\text{ini}}) \leq\ & 2 \cdot \boldsymbol{\mu}^{\text{xx}}(q_{\text{ro}}) \cdot \mathbf{Adv}^{\text{xx}}_{\mathbb{G},g,p}(A_{\text{xx}}) \\ & + \frac{2 \cdot q_{\text{rn}}(q_{\text{rn}} + q_{\text{ro}})}{2^{sl}} + \frac{2 \cdot \boldsymbol{\delta}^{\text{xx}}(q_{\text{ro}})}{p} \ .\end{aligned} \tag{19}$$

where $q_{\text{rn}} = \mathrm{Q}^{\text{Run}}(A_{\text{ini}})$ and $q_{\text{ro}} = \mathrm{Q}^{\text{RO}}(A_{\text{ini}})$. Further

$$\boldsymbol{\mu}^{\text{xx}}(q_{\text{ro}}) = \begin{cases} q_{\text{ro}} & \text{if xx} \in \{\text{cdh}, \text{cdh-muc}\} \\ 1 & \text{if xx} \in \{\text{v-cdh}, \text{v-cdh-muc}, \text{ddh}\} \end{cases} \tag{20}$$

and

$$\boldsymbol{\delta}^{\text{xx}}(q_{\text{ro}}) = \begin{cases} 0 & \text{if xx} \in \{\text{cdh}, \text{v-cdh}, \text{cdh-muc}, \text{v-cdh-muc}\} \\ q_{\text{ro}} + 1 & \text{if xx} = \text{ddh} \end{cases}$$

Protocol Π	Problem P	$\mathbf{B}(\epsilon', \{q_{\text{ro}}, q_{\text{rn}}\})$	Number of queries			
			NewKey	NewBase	DDHO	CDHO
2H-DH PSI	CDH	$4 \cdot (q_{\text{ro}}^2 q_{\text{rn}} \cdot \epsilon' + \alpha_c)$	–	–	–	–
	V-CDH	$4 \cdot (q_{\text{ro}} q_{\text{rn}} \cdot \epsilon' + \alpha_c)$	–	–	q_{ro}	–
	CDH-MUC	$4 \cdot (q_{\text{ro}} \cdot \epsilon' + \alpha_c)$	q_{rn}	q_{ro}	–	M
	V-CDH-MUC	$4 \cdot (\epsilon' + \alpha_c)$	q_{rn}	q_{ro}	q_{ro}	M
	DDH	$4 \cdot (\epsilon' + \alpha_d)$	–	–	–	–
Salted DH PSI	CDH	$2 \cdot (q_{\text{ro}} \cdot \epsilon' + \beta_c)$	–	–	–	–
	V-CDH	$2 \cdot (\epsilon' + \beta_c)$	–	–	q_{ro}	–
	CDH-MUC	$2 \cdot (q_{\text{ro}} \cdot \epsilon' + \beta_c)$	1	1	–	0
	V-CDH-MUC	$2 \cdot (\epsilon' + \beta_c)$	1	1	q_{ro}	0
	DDH	$2 \cdot (\epsilon' + \beta_d)$	–	–	–	–

Figure 15: **Our results for PSI.** We compare our results for the classical DH-PSI protocol with those for our new Salted DH PSI protocol. For different choices of the assumed-hard problem P, we show the bound $\mathbf{B}(\epsilon', \{q_{\text{ro}}, q_{\text{rn}}\})$ on the advantage of an adversary A in game $\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,2}$ as a function of the advantage $\epsilon' = \mathbf{Adv}^{\text{P}}_{\mathbb{G}}(A')$ of the constructed adversary A' in solving problem P in group $\mathbb{G}$. We also show the query profile of A'. Here, $q_{\text{ro}} = \text{Q}^{\text{RO}}(A)$ and $q_{\text{rn}} = \text{Q}^{\text{Run}}(A)$. We have set $\alpha_c = (q_{\text{ro}} \cdot q_{\text{rn}})/p$ and $\alpha_d = (q_{\text{ro}} \cdot q_{\text{rn}} + q_{\text{ro}} + 1)/p$. With sl being the salt length in the Salted DH PSI protocol we have also set $\beta_c = q_{\text{rn}} \cdot (q_{\text{rn}} + q_{\text{ro}}) \cdot 2^{-sl}$ and $\beta_d = q_{\text{rn}} \cdot (q_{\text{rn}} + q_{\text{ro}}) \cdot 2^{-sl} + (q_{\text{ro}} + 1)/p$. By M we denote an upper bound on the sum, across all Run queries of adversary A, of the sizes of the sets in these queries. Adversaries A and A' have about the same running time. A dash (–) means that the oracle is not present for P.

---

with resources

$$\begin{aligned}
\text{Q}^{\text{NewKey}}(A_{\text{cdh-muc}}) &= \text{Q}^{\text{NewKey}}(A_{\text{v-cdh-muc}}) = 1 \\
\text{Q}^{\text{NewBase}}(A_{\text{cdh-muc}}) &= \text{Q}^{\text{NewBase}}(A_{\text{v-cdh-muc}}) = 1 \ , \\
\text{Q}^{\text{CDHO}}(A_{\text{cdh-muc}}) &= \text{Q}^{\text{CDHO}}(A_{\text{v-cdh-muc}}) = 0 \ , \\
\text{Q}^{\text{DDHO}}(A_{\text{v-cdh}}) &= \text{Q}^{\text{DDHO}}(A_{\text{v-cdh-muc}}) \leq \text{Q}^{\text{RO}}(A_{\text{ini}}) \ .
\end{aligned}$$

The running times of $A_{\text{v-cdh-muc}}$ and $A_{\text{cdh-muc}}$ are about that of A_{ini}. The adversaries $A_{\text{v-cdh}}$ and $A_{\text{v-cdh}}$ perform an additional $\text{Q}^{\text{Run}}(A_{\text{ini}}) + M + \text{Q}^{\text{RO}}(A_{\text{ini}})$ group exponentiations and A_{ddh} performs an additional $\text{Q}^{\text{Run}}(A_{\text{ini}}) + 2 \cdot M + 2 \cdot \text{Q}^{\text{RO}}(A_{\text{ini}})$ group exponentiations.

Acknowledgments

We thank Dan Boneh for discussions in the context of the Apple PSI protocol. We thank Matilda Backendal and Sue Hong for comments and corrections on a prior draft. We thank the reviewers of Crypto 2024 and Asiacrypt 2024 for their reviews and feedback.

References

[1] Abdalla, M., Bellare, M., Rogaway, P.: The oracle Diffie-Hellman assumptions and an analysis of DHIES. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 143–158. Springer, Heidelberg (Apr 2001) (Cited on page 8, 23.)

[2] Agarap, A.F.: Deep learning using rectified linear units (relu) (2019) (Cited on page 39.)

[3] Agrawal, S., Agrawal, S., Prabhakaran, M.: Cryptographic agents: Towards a unified theory of computing on encrypted data. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 501–531. Springer, Heidelberg (Apr 2015) (Cited on page 8.)

[4] Apple: Password monitoring. https://support.apple.com/guide/security/password-monitoring-sec78e79fc3b/web (Feb 2021) (Cited on page 7.)

[5] Baldi, P., Baronio, R., De Cristofaro, E., Gasti, P., Tsudik, G.: Countering GATTACA: efficient and secure testing of fully-sequenced human genomes. In: Chen, Y., Danezis, G., Shmatikov, V. (eds.) ACM CCS 2011. pp. 691–702. ACM Press (Oct 2011) (Cited on page 7.)

[6] Bellare, M.: A concrete-security analysis of the apple psi protocol. https://www.apple.com/child-safety/pdf/Alternative_Security_Proof_of_Apple_PSI_System_Mihir_Bellare.pdf (July 2021) (Cited on page 8.)

[7] Bellare, M., Canetti, R., Krawczyk, H.: Pseudorandom functions revisited: The cascade construction and its concrete security. In: 37th FOCS. pp. 514–523. IEEE Computer Society Press (Oct 1996) (Cited on page 11.)

[8] Bellare, M., Hoang, V.T., Keelveedhi, S., Rogaway, P.: Efficient garbling from a fixed-key blockcipher. In: 2013 IEEE Symposium on Security and Privacy. pp. 478–492. IEEE Computer Society Press (May 2013) (Cited on page 5.)

[9] Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) ACM CCS 2012. pp. 784–796. ACM Press (Oct 2012) (Cited on page 8.)

[10] Bellare, M., Kilian, J., Rogaway, P.: The security of cipher block chaining. In: Desmedt, Y. (ed.) CRYPTO'94. LNCS, vol. 839, pp. 341–358. Springer, Heidelberg (Aug 1994) (Cited on page 3.)

[11] Bellare, M., Namprempre, C., Pointcheval, D., Semanko, M.: The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. Journal of Cryptology 16(3), 185–215 (Jun 2003) (Cited on page 7.)

[12] Bellare, M., O'Neill, A.: Semantically-secure functional encryption: Possibility results, impossibility results and the quest for a general definition. In: Abdalla, M., Nita-Rotaru, C., Dahab, R. (eds.) CANS 13. LNCS, vol. 8257, pp. 218–234. Springer, Heidelberg (Nov 2013) (Cited on page 9.)

[13] Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) ACM CCS 93. pp. 62–73. ACM Press (Nov 1993) (Cited on page 6, 9, 10.)

[14] Bellare, M., Rogaway, P.: The exact security of digital signatures: How to sign with RSA and Rabin. In: Maurer, U.M. (ed.) EUROCRYPT'96. LNCS, vol. 1070, pp. 399–416. Springer, Heidelberg (May 1996) (Cited on page 8.)

[15] Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 409–426. Springer, Heidelberg (May / Jun 2006) (Cited on page 10, 46, 51, 52.)

[16] Blum, M., Micali, S.: How to generate cryptographically strong sequences of pseudorandom bits. SIAM Journal on Computing 13(4), 850–864 (1984) (Cited on page 3.)

[17] Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In: Desmedt, Y. (ed.) PKC 2003. LNCS, vol. 2567, pp. 31–46. Springer, Heidelberg (Jan 2003) (Cited on page 7.)

[18] Boneh, D., Sahai, A., Waters, B.: Functional encryption: Definitions and challenges. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 253–273. Springer, Heidelberg (Mar 2011) (Cited on page 9.)

[19] Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press (Oct 2001) (Cited on page 5, 9, 12, 14.)

[20] Chase, M., Miao, P.: Private set intersection in the internet setting from lightweight oblivious PRF. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 34–63. Springer, Heidelberg (Aug 2020) (Cited on page 6.)

[21] Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 1243–1255. ACM Press (Oct / Nov 2017) (Cited on page 6.)

[22] Davidson, A., Goldberg, I., Sullivan, N., Tankersley, G., Valsorda, F.: Privacy pass: Bypassing internet challenges anonymously. PoPETs 2018(3), 164–180 (Jul 2018) (Cited on page 7, 26.)

[23] Everspaugh, A., Chatterjee, R., Scott, S., Juels, A., Ristenpart, T.: The pythia PRF service. In: Jung, J., Holz, T. (eds.) USENIX Security 2015. pp. 547–562. USENIX Association (Aug 2015) (Cited on page 7, 26.)

[24] Feige, U., Lapidot, D., Shamir, A.: Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In: 31st FOCS. pp. 308–317. IEEE Computer Society Press (Oct 1990) (Cited on page 8.)

[25] Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 303–324. Springer, Heidelberg (Feb 2005) (Cited on page 6, 7.)

[26] Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 1–19. Springer, Heidelberg (May 2004) (Cited on page 5, 7, 19, 20.)

[27] Goldreich, O.: Foundations of Cryptography: Basic Tools, vol. 1. Cambridge University Press, Cambridge, UK (2001) (Cited on page 12.)

[28] Goldreich, O.: Foundations of Cryptography: Basic Applications, vol. 2. Cambridge University Press, Cambridge, UK (2004) (Cited on page 12.)

[29] Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. Journal of the ACM 33(4), 792–807 (Oct 1986) (Cited on page 11.)

[30] Goldwasser, S., Micali, S.: Probabilistic encryption. Journal of Computer and System Sciences 28(2), 270–299 (1984) (Cited on page 3.)

[31] Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on Computing 18(1), 186–208 (1989) (Cited on page 8.)

[32] Hazay, C., Lindell, Y.: Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 155–175. Springer, Heidelberg (Mar 2008) (Cited on page 3, 7, 20, 22.)

[33] Hunt, T., Hunt, C., Siguroarson, S.: Have I been pwned? https://haveibeenpwned.com/ (Cited on page 7.)

[34] Ion, M., Kreuter, B., Nergiz, A.E., Patel, S., Raykova, M., Saxena, S., Seth, K., Shanahan, D., Yung, M.: On deploying secure computing commercially: Private intersection-sum protocols and their business applications. Cryptology ePrint Archive, Report 2019/723 (2019), https://eprint.iacr.org/2019/723 (Cited on page 7.)

[35] Ion, M., Kreuter, B., Nergiz, E., Patel, S., Saxena, S., Seth, K., Shanahan, D., Yung, M.: Private intersection-sum protocol with applications to attributing aggregate ad conversions. Cryptology ePrint Archive, Report 2017/738 (2017), https://eprint.iacr.org/2017/738 (Cited on page 7.)

[36] Jarecki, S., Kiayias, A., Krawczyk, H.: Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 233–253. Springer, Heidelberg (Dec 2014) (Cited on page 3, 4, 7, 11, 26.)

[37] Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part III. LNCS, vol. 10822, pp. 456–486. Springer, Heidelberg (Apr / May 2018) (Cited on page 7, 26.)

[38] Jarecki, S., Liu, X.: Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 577–594. Springer, Heidelberg (Mar 2009) (Cited on page 7.)

[39] Juvekar, C., Vaikuntanathan, V., Chandrakasan, A.: GAZELLE: A low latency framework for secure neural network inference. In: Enck, W., Felt, A.P. (eds.) USENIX Security 2018. pp. 1651–1669. USENIX Association (Aug 2018) (Cited on page 5, 38.)

[40] Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious PRF with applications to private set intersection. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 818–829. ACM Press (Oct 2016) (Cited on page 6.)

[41] Kolesnikov, V., Matania, N., Pinkas, B., Rosulek, M., Trieu, N.: Practical multi-party private set intersection from symmetric-key techniques. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 1257–1272. ACM Press (Oct / Nov 2017) (Cited on page 6.)

[42] Li, L., Pal, B., Ali, J., Sullivan, N., Chatterjee, R., Ristenpart, T.: Protocols for checking compromised credentials. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 1387–1403. ACM Press (Nov 2019) (Cited on page 7.)

[43] Lindell, Y.: How to simulate it - A tutorial on the simulation proof technique. Cryptology ePrint Archive, Report 2016/046 (2016), https://eprint.iacr.org/2016/046 (Cited on page 5, 12, 14.)

[44] Marlinspike, M.: The difficulty of private contact discovery. https://signal.org/blog/contact-discovery/ (Jan 2014) (Cited on page 7.)

[45] Mezzour, G., Perrig, A., Gligor, V.D., Papadimitratos, P.: Privacy-preserving relationship path discovery in social networks. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 09. LNCS, vol. 5888, pp. 189–208. Springer, Heidelberg (Dec 2009) (Cited on page 7.)

[46] Naor, M., Reingold, O.: Number-theoretic constructions of efficient pseudo-random functions. In: 38th FOCS. pp. 458–467. IEEE Computer Society Press (Oct 1997) (Cited on page 7.)

[47] Narayanan, A., Thiagarajan, N., Lakhani, M., Hamburg, M., Boneh, D.: Location privacy via private proximity testing. In: NDSS 2011. The Internet Society (Feb 2011) (Cited on page 7.)

[48] Okamoto, T., Pointcheval, D.: The gap-problems: A new class of problems for the security of cryptographic schemes. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 104–118. Springer, Heidelberg (Feb 2001) (Cited on page 7, 8, 23.)

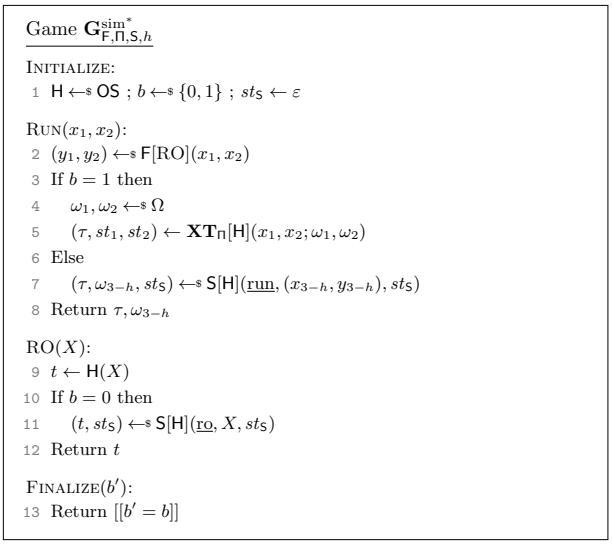
[49] O'Neill, A.: Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556 (2010), https://eprint.iacr.org/2010/556 (Cited on page 9.)

[50] Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: SpOT-light: Lightweight private set intersection from sparse OT extension. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 401–431. Springer, Heidelberg (Aug 2019) (Cited on page 6.)

[51] Pinkas, B., Schneider, T., Segev, G., Zohner, M.: Phasing: Private set intersection using permutation-based hashing. In: Jung, J., Holz, T. (eds.) USENIX Security 2015. pp. 515–530. USENIX Association (Aug 2015) (Cited on page 6.)

[52] Rabin, M.O.: How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187 (2005), https://eprint.iacr.org/2005/187 (Cited on page 5, 38.)

[53] Rajan, A., Qin, L., Archer, D.W., Boneh, D., Lepoint, T., Varia, M.: Callisto: A cryptographic approach to detecting serial perpetrators of sexual misconduct. In: Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies. pp. 1–4 (2018) (Cited on page 7.)



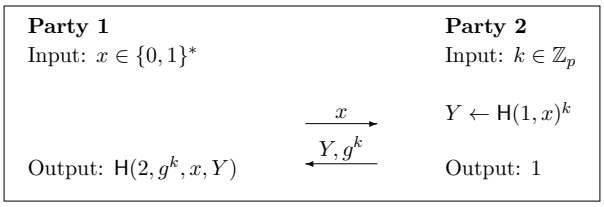


Figure 16: **Top:** Game defining SIM* security for protocol Π for functionality F, where $h \in \{1,2\}$ is the honest party and S is the simulator. **Bottom:** The "bad" protocol Π realizing the functionality $\mathsf{F}^{\mathrm{oprf}}_{\mathsf{2HDH}}$.

[54] Thomas, K., Pullman, J., Yeo, K., Raghunathan, A., Kelley, P.G., Invernizzi, L., Benko, B., Pietraszek, T., Patel, S., Boneh, D., Bursztein, E.: Protecting accounts from credential stuffing with password breach alerting. In: Heninger, N., Traynor, P. (eds.) USENIX Security 2019. pp. 1556–1571. USENIX Association (Aug 2019) (Cited on page 7.)

[55] Tyagi, N., Celi, S., Ristenpart, T., Sullivan, N., Tessaro, S., Wood, C.A.: A fast and simple partially oblivious PRF, with applications. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 674–705. Springer, Heidelberg (May / Jun 2022) (Cited on page 4, 7, 22.)

[56] Yao, A.C.C.: Protocols for secure computations (extended abstract). In: 23rd FOCS. pp. 160–164. IEEE Computer Society Press (Nov 1982) (Cited on page 3.)

A The subtle point about SIM: Unsoundness of SIM*

In Section 3.1 we discussed a subtlety in how one defines simulation-based security in the programmable-ROM. The general understanding and intent of the definition is that random-oracle queries are answered by the simulator. The subtlety is that, while this is fine for queries made by the adversary and protocol, it is *not* fine for queries made by the functionality. (Accordingly, functionality queries to the RO are answered honestly in our SIM game of Figure 1.) Here we treat the example showing this in more detail.

THE SIM* DEFINITION. We start by specifying the definition that we see as the first and natural choice yet will show is incorrect. We call it SIM*. The game $\mathbf{G}^{\mathrm{sim}^*}_{\mathsf{F},\Pi,\mathsf{S},h}$ is shown in Fig 16. The only change from the $\mathbf{G}^{\mathrm{sim}}_{\mathsf{F},\Pi,\mathsf{S},h}$ game is that, at line 2, the oracle provided to the functionality F is RO,

not H. So when the challenge bit b is 0, the RO queries of the functionality will be answered by the simulator, just like all other RO queries. For a protocol Π, functionality F, simulator S and honest party $h \in \{1,2\}$ we let $\mathbf{Adv}^{\mathrm{sim}^*}_{\mathsf{F},\Pi,\mathsf{S},h}(A_{\mathrm{sim}^*}) = 2\Pr[\mathbf{G}^{\mathrm{sim}^*}_{\mathsf{F},\Pi,\mathsf{S},h}(A_{\mathrm{sim}^*})] - 1$ be the advantage of an adversary A_{sim^*}.

THE BAD PROTOCOL. Let $\mathbb{G} = \langle g \rangle$ be a group of prime order p with generator g. Let $\mathsf{2HDH}\colon [\mathsf{OS}] \times \mathbb{Z}_p \times \{0,1\}^* \to \{0,1\}^\ell$ be the 2H-DH PRF associated to $\mathbb{G}$ and $\ell \geq 1$ as per Figure 3. Let $\mathsf{F} = \mathsf{F}^{\mathrm{oprf}}_{\mathsf{2HDH}}$ then be the OPRF functionality associated to $\mathsf{2HDH}$ as per Figure 3. The input to the client is in general a vector over $\{0,1\}^*$, but we will need to consider only a single input $x \in \{0,1\}^*$, and see the functionality thus as $\mathsf{F}^{\mathrm{oprf}}_{\mathsf{2HDH}}[\mathsf{H}](x,k) = (\mathsf{H}(2, g^k, x, \mathsf{H}(1,x)^k), 1)$. (The output of party 2 is the length of the vector, here 1.) Consider the protocol Π, shown in Figure 16, which realizes the $\mathsf{F}^{\mathrm{oprf}}_{\mathsf{2HDH}}$ functionality. Party 1 sends its input x in the clear and party 2 uses it to calculate $Y \leftarrow \mathsf{H}(1,x)^k$. Party 2 then sends Y, g^k to party 1 and outputs 1. Finally, party 1 calculates $\mathsf{H}(2, g^k, x, Y)$ as its output. Since party 1 sends its input in the clear, Π clearly does not provide input privacy for party 1 and should be deemed insecure. However, in the following theorem we show that Π is SIM* secure for $\mathsf{F}^{\mathrm{oprf}}_{\mathsf{2HDH}}$ and honest party $h = 1$.

Theorem A.1 *Let $\mathbb{G}, \ell, \mathsf{2HDH}, \mathsf{F} = \mathsf{F}^{\mathrm{oprf}}_{\mathsf{2HDH}}$ be as above, and let Π be the protocol of Figure 16 that computes F. We can construct a linear-time simulator S such that for all adversaries A_{sim^*} playing game $\mathbf{G}^{\mathrm{sim}^*}_{\mathsf{F},\Pi,\mathsf{S},1}$ we have:*

$$\mathbf{Adv}^{\mathrm{sim}^*}_{\mathsf{F},\Pi,\mathsf{S},1}(A_{\mathrm{sim}^*}) = 0\ . \tag{21}$$

We note that Π is *not* SIM-secure, meaning our SIM definition correctly excludes it.

Proof of Theorem A.1: We first specify how S operates in its ro role, meaning when answering RO queries. Here it has an input X for H, and takes its current state st_{S}, and does the following:

$\mathsf{S}[\mathsf{H}](\underline{\mathrm{ro}}, X, st_{\mathsf{S}})$:
1. $(n, X') \leftarrow X$ // Parse X in this way
2. If $(n = 1)$ then $st_{\mathsf{S}} \leftarrow X'$ // Store X' in the state
3. Return $\mathsf{H}(X)$

Now let us explain. We know that X has the form (n, X') with $n \in \{1,2\}$. If $n = 1$ then the simulator stores X', which will be a string, in its state. The reply to the RO query is, regardless, always given honestly, via the function H to which S has oracle access.

Queries to RO, that are answered as above when $b = 0$, will come from the adversary, the protocol, and also the functionality at line 2. The simulator is answering them honestly, and simply remembering the most recent query to $\mathsf{H}(1,\cdot)$. (The salient point is that at the time line 7 is executed, this query will have been from line 2.) Now we define how the simulator operates in its run role. Its input here is $(k,1)$ —key k and output 1 from the functionality— and state st_{S}. It does the following:

$\mathsf{S}[\mathsf{H}](\underline{\mathrm{run}}, (k,1), st_{\mathsf{S}})$:
1. $x \leftarrow st_{\mathsf{S}}$ // Recover string $x \in \{0,1\}^*$ stored in the state
2. $Y \leftarrow \mathsf{H}(1,x)^k$; $\tau \leftarrow (x, (Y, g^k))$
3. Return (τ, ε)

Suppose adversary A_{sim^*} queries RUN with inputs x, k. At line 2 of the game the functionality F is evaluated on these inputs. As per its definition, it would query the random oracle RO first with input $(1, x)$ to get an output Z and next with input $(2, g^k, x, Z^k)$. Let b be the challenge bit of the game. If $b = 0$ the RO queries made by the functionality would be answered by S running in its ro role. So it would store x in its state.

Note that the game calls S in run mode right after F queries RO oracle at line 2. Hence, the last input that S stores in its state would always be the input of party 1 for which it needs to generate the transcript. As it has inputs for both parties, S just follows Π to accurately construct a transcript which is then distributed identically to the one that Π outputs. So $\mathbf{Adv}^{\mathrm{sim}^*}_{\mathsf{F},\Pi,\mathsf{S},1}(A_{\mathrm{sim}^*}) = 0$. ∎

B Proof of Theorem 3.2

Proof of Theorem 3.2: Let S be any simulator. Adversary A_{sim} is playing game $\mathbf{G}^{\mathrm{sim}}_{\mathsf{F},\Pi,\mathsf{S},h}$. Adversary A_{sim} picks a challenge bit $c \leftarrow\!\!\$\ \{0,1\}$ and runs A_{ini}. When A_{ini} makes query $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F},\Pi,h}$. $\textsc{Run}(x_0, x_1, x)$, adversary A_{sim} does the following:

1. $x_{3-h,0} \leftarrow x$; $x_{3-h,1} \leftarrow x$; $x_{h,0} \leftarrow x_0$; $x_{h,1} \leftarrow x_1$
2. $(y_{1,0}, y_{2,0}) \leftarrow \mathsf{F}[\mathrm{RO}](x_{1,0}, x_{2,0})$; $(y_{1,1}, y_{2,1}) \leftarrow \mathsf{F}[\mathrm{RO}](x_{1,1}, x_{2,1})$
3. If $(y_{3-h,0} \neq y_{3-h,1})$ then return $\perp$
4. $(\tau, \omega_{3-h}) \leftarrow\!\!\$\ \mathbf{G}^{\mathrm{sim}}_{\mathsf{F},\Pi,\mathsf{S},h}.\textsc{Run}(x_{1,c}, x_{2,c})$
5. Return (τ, ω_{3-h})

Above, at line 2, RO denotes $\mathbf{G}^{\mathrm{sim}}_{\mathsf{F},\Pi,\mathsf{S},h}.\mathrm{RO}$, the random oracle provided to A_{sim} by its own game. When A_{ini} queries $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F},\Pi,h}.\mathrm{RO}(X)$, adversary A_{sim} returns $Y \leftarrow \mathbf{G}^{\mathrm{sim}}_{\mathsf{F},\Pi,\mathsf{S},h}.\mathrm{RO}(X)$ to A_{ini} as the reply. Eventually A_{ini} halts with output guess $c' \in \{0,1\}$. Adversary A_{sim} lets $b' \leftarrow [[c' = c]]$ and returns b' as its own output. For the analysis, let b denote the challenge bit chosen at line 1 on the right of Figure 5. We observe that

$$\Pr\left[\, b' = 1 \mid b = 1 \,\right] = \Pr\left[\, \mathbf{G}^{\mathrm{ini}}_{\mathsf{F},\Pi,h}(A_{\mathrm{ini}}) \,\right] = \frac{1}{2} + \frac{1}{2} \cdot \mathbf{Adv}^{\mathrm{ini}}_{\mathsf{F},\Pi,h}(A_{\mathrm{ini}}) \ .$$

Now when $b = 0$, the input provided to the simulator at line 7 on the right of Figure 5 is $x_{3-h,c}, y_{3-h,c}$. But $x_{3-h,c} = x$ (line 1 above) and $y_{3-h,0} = y_{3-h,1}$ (due to line 3 above) so what is provided to the simulator does not depend on c. Hence

$$\Pr\left[\, b' = 1 \mid b = 0 \,\right] = \frac{1}{2} \ .$$

Thus

$$\begin{aligned} \mathbf{Adv}^{\mathrm{sim}}_{\mathsf{F},\Pi,\mathsf{S},h}(A_{\mathrm{sim}}) &= \Pr\left[\, b' = 1 \mid b = 1 \,\right] - \Pr\left[\, b' = 1 \mid b = 0 \,\right] \\ &= \frac{1}{2} + \frac{1}{2} \cdot \mathbf{Adv}^{\mathrm{ini}}_{\mathsf{F},\Pi,h}(A_{\mathrm{ini}}) - \frac{1}{2} = \frac{1}{2} \cdot \mathbf{Adv}^{\mathrm{ini}}_{\mathsf{F},\Pi,h}(A_{\mathrm{ini}}) \ , \end{aligned}$$

which yields Eq. (2). ∎

C Proof of Theorem 3.3

Proof of Theorem 3.3: In protocol Π shown on the right side of Figure 6, party 1 sends its input x in the clear to party 2. Party 2 then calculates its output as g^x. As we can see Π correctly realizes the functionality F.

Now any adversary $A_{\rm ini}$, playing game $\mathbf{G}^{\rm ini}_{\mathsf{F},\Pi,1}$, needs to find two values $x_0, x_1 \in \mathbb{Z}_p$ such that $g^{x_0} = g^{x_1}$ to go past the check at line 5 in the left panel of Figure 5. Since g is a generator for $\mathbb{G}$, this can only happen if $x_0 = x_1$. So either $A_{\rm ini}$ sends the same inputs for the honest party or it gets $\perp$ as output from its $\mathbf{G}^{\rm ini}_{\mathsf{F},\Pi,1}$.RUN oracle queries. In both cases it gets no information about the challenge bit. Therefore, we have $\Pr[\mathbf{G}^{\rm ini}_{\mathsf{F},\Pi,1}(A_{\rm ini})] = 1/2$, which gives $\mathbf{Adv}^{\rm ini}_{\mathsf{F},\Pi,1}(A_{\rm ini}) = 0$. This establishes Eq. (3) and proves the first part of the theorem.

For the second part, we define adversary $A_{\rm sim}$, playing game $\mathbf{G}^{\rm sim}_{\mathsf{F},\Pi,\mathsf{S},1}$, as follows. It picks $x \leftarrow\!\!\$\ \mathbb{Z}_p$ and makes a RUN(x,ε) query to get a transcript τ. If $\tau = x$, it sets $b' \leftarrow 1$ else $b' \leftarrow 0$. It then returns b' as its guess for the challenge bit b of game $\mathbf{G}^{\rm sim}_{\mathsf{F},\Pi,\mathsf{S},1}$. Since the transcript created by Π would be x, we have

$$\Pr\left[\, b' = 1 \mid b = 1 \,\right] = 1 \ .$$

When $b = 0$, the transcript τ is produced by the simulator S given g^x but not x. Now the idea is that either $\tau = x$, in which case we will be able to solve the DL problem via $A_{\rm dl}$, or $\tau \neq x$, in which case $A_{\rm sim}$ will correctly determine the challenge bit. To formalize this and complete the proof, we construct adversary $A_{\rm dl}$, playing game $\mathbf{G}^{\rm dl}_{\mathbb{G},g,p}$, as follows. It gets input $X = g^x \in \mathbb{G}$. It lets $(\tau, \varepsilon, st_{\mathsf{S}}) \leftarrow\!\!\$\ \mathsf{S}(\underline{\rm run}, (\varepsilon, X), \varepsilon)$, meaning it runs the simulator in its $\underline{\rm run}$ role, with input ε and output X for the corrupted party, and ε as simulator state, to get back transcript τ, coins ε and updated state st_{S}. It then returns τ as its output for game $\mathbf{G}^{\rm dl}_{\mathbb{G},g,p}$. Clearly, $A_{\rm dl}$ wins if $\tau = x$. Thus

$$\Pr\left[\, b' = 1 \mid b = 0 \,\right] = \mathbf{Adv}^{\rm dl}_{\mathbb{G},g,p}(A_{\rm dl}) \ .$$

Thus

$$\begin{aligned}\mathbf{Adv}^{\rm sim}_{\mathsf{F},\Pi,\mathsf{S},1}(A_{\rm sim}) &= \Pr\left[\, b' = 1 \mid b = 1 \,\right] - \Pr\left[\, b' = 1 \mid b = 0 \,\right] \\ &= 1 - \mathbf{Adv}^{\rm dl}_{\mathbb{G},g,p}(A_{\rm dl}) \ ,\end{aligned}$$

which is Eq. (4). ∎

D Proof of Theorem 3.4

Proof of Theorem 3.4: We define simulator S, starting with its $\underline{\rm run}$ role. Here it receives input (x_{3-h}, y_{3-h}) and its current state st_{S}, and has access to an oracle $\mathsf{H} \in \mathsf{OS}$. It does the following:

$\underline{\mathsf{S}[\mathsf{H}](\underline{\rm run}, (x_{3-h}, y_{3-h}), st_{\mathsf{S}})}$:
1. $x'_h \leftarrow\!\!\$\ \mathrm{IA}[\mathsf{H}](x_{3-h}, y_{3-h})$; $\omega_1, \omega_2 \leftarrow\!\!\$\ \Omega$; $x'_{3-h} \leftarrow x_{3-h}$
2. $(\tau, st_1, st_2) \leftarrow \mathbf{XT}_{\Pi}[\mathsf{H}](x'_1, x'_2; \omega_1, \omega_2)$
3. Return $(\tau, \omega_{3-h}, st_{\mathsf{S}})$

That is, S honestly executes the protocol using its provided input for the corrupted party and an input x'_h for the honest party that is obtained via the inverter. It returns the transcript, and coins

for the corrupted party. Since S is for the SIM-np game, it has no ro role, so the above completes its description. Its running time is dominated by the time to run IA and to execute Π.

Given adversary A_{snp} playing game $\mathbf{G}^{\text{sim-np}}_{\mathsf{F},\Pi,\mathsf{S},h}$, we construct adversary A_{ini} playing game $\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,h}$ as follows. Adversary A_{ini} runs adversary A_{snp}. When A_{snp} makes a query $\mathbf{G}^{\text{sim-np}}_{\mathsf{F},\Pi,\mathsf{S},h}.\text{RUN}(x_1, x_2)$, adversary A_{ini} does the following:

1. $(y_1, y_2) \leftarrow \mathsf{F}[\mathsf{H}](x_1, x_2)$
2. $x'_h \leftarrow\!\!\$\ \text{IA}[\mathsf{H}](x_{3-h}, y_{3-h})$; $x'_{3-h} \leftarrow x_{3-h}$
3. $(\tau, \omega) \leftarrow \mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,h}.\text{RUN}(x'_h, x_h, x_{3-h})$
4. Return (τ, ω) to A_{snp} as the answer to its query

When A_{snp} queries $\mathbf{G}^{\text{sim-np}}_{\mathsf{F},\Pi,\mathsf{S},h}.\text{RO}(X)$, adversary A_{ini} lets $Y \leftarrow \mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,h}.\text{RO}(X)$ and returns Y to A_{snp}. Finally, A_{snp} outputs a bit b' and A_{ini} returns the same bit. Let b be the challenge bit of game $\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,h}$. When $b = 1$, the call to oracle $\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,h}.\text{RUN}(x'_h, x_h, x_{3-h})$ gives the output of running Π on inputs x_1, x_2, which is what A_{snp} would receive in the real game from $\mathbf{G}^{\text{sim-np}}_{\mathsf{F},\Pi,\mathsf{S},h}.\text{RUN}(x_1, x_2)$. In the other case when $b = 0$, the call to oracle $\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,h}.\text{RUN}(x'_h, x_h, x_{3-h})$ gives the output of running Π on inputs x'_1, x'_2, which is what A_{snp} would receive in the ideal game from $\mathbf{G}^{\text{sim-np}}_{\mathsf{F},\Pi,\mathsf{S},h}.\text{RUN}(x_1, x_2)$ due to the way S works in its run role. With these observations we get

$$\Pr[\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,h}(A_{\text{ini}})] = \Pr[\mathbf{G}^{\text{sim-np}}_{\mathsf{F},\Pi,\mathsf{S},h}(A_{\text{snp}})] \ ,$$

and thus

$$\mathbf{Adv}^{\text{sim-np}}_{\mathsf{F},\Pi,\mathsf{S},h}(A_{\text{snp}}) = \mathbf{Adv}^{\text{ini}}_{\mathsf{F},\Pi,h}(A_{\text{ini}}) \ ,$$

which yields Eq. (5). Note that A_{ini} runs A_{snp} and uses IA but running time is maintained because S also runs IA and our convention is that we measure the time for the execution of the game with the adversary. ∎

E More Invertible Functionalities

In Section 3.3 we showed that the PSI functionality and its variants are invertible. Here we show some more functionalities with practical importance that are invertible. In particular, we show invertibility for the widely used *Oblivious Transfer* (OT) [52] functionality and a simpler version of secure inferencing functionality for neural networks as described in [39].

OT FUNCTIONALITY. The 1-out-of-n OT functionality $\mathsf{F}^{\text{ot}}_n \colon \{1, \ldots, n\} \times (\{0,1\}^*)^n \to \{0,1\}^* \times \{\varepsilon\}$ is evaluated for inputs $(c, \boldsymbol{x})$ where $c \in \{1, \ldots, n\}$ and $\boldsymbol{x} \in (\{0,1\}^*)^n$ as $\mathsf{F}^{\text{ot}}_n(c, \boldsymbol{x}) = (\boldsymbol{x}[c], \varepsilon)$. The idea is that party 1 holds a value representing an index and party 2 holds a vector of elements in $\{0,1\}^*$. The functionality allows party 1 to get the value indexed by its input from party 2.

INVERTERS FOR OT. To show invertibility of OT functionality we present the inverter $\text{IA}^{\text{ot}}_{n,h}$ for honest party $h \in \{1, 2\}$ in the left part of Figure 17.

The inverter $\text{IA}^{\text{ot}}_{n,1}$ gets as inputs the input of party 2 to F^{ot}_n which is a vector $\boldsymbol{x}$ and the output of party 2 from F^{ot}_n which is always ε irrespective of the input of party 1. Therefore, the inverter can return any value from the set $\{1, \ldots, n\}$ to be a correct inverter which is what is done in Figure 17 (which returns 1).

When $h = 2$ we have the inverter $\text{IA}^{\text{ot}}_{n,2}$ which gets an index $c \in \{1, \ldots, n\}$ and a string $y \in \{0,1\}^*$ as input where $y = \mathsf{F}^{\text{ot}}_n(c, \boldsymbol{x})[1]$ for some $\boldsymbol{x}$ which is the input of party 2 unknown to

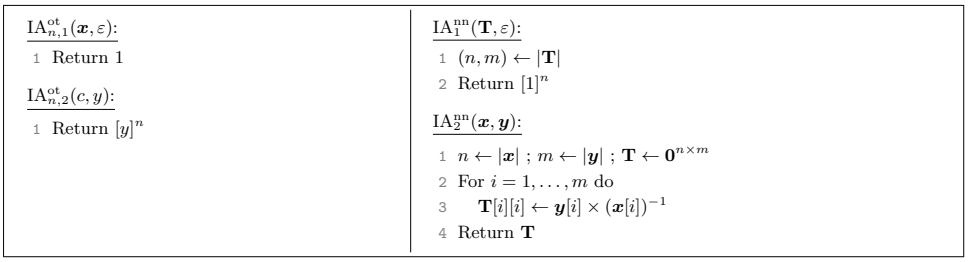


Figure 17: **Left:** Inverter for OT functionality for $h = 1$. **Right:** Inverter for $\mathsf{F}^{\mathrm{cnn}}$ functionality for $h = 2$.

party 1. For the inverter to be correct, it suffices that it returns a vector $\boldsymbol{x}'$ with $\boldsymbol{x}'[c] = y$. This is what is done in $\mathrm{IA}^{\mathrm{ot}}_{n,2}$ which returns a vector with all elements as y.

Clearly, both the inverters are efficient.

We use matrices below for which we use the following notation. If $\boldsymbol{X}$ is an n by m matrix then we let $|\boldsymbol{X}| = (n, m)$. By $\boldsymbol{X}[i][j]$ we denote the entry in row $i \in [1..n]$ and column $j \in [1..m]$. We denote the n-by-m zero matrix by $\mathbf{0}^{n\times m}$. By $(c)^n$ we denote the n-vector all of whose entries are c.

Secure Inferencing functionality. Let n, m be natural numbers. We define a simple neural network inferencing function, $\mathsf{Q}^{\mathrm{nn}}\colon \mathbb{R}^n \times \mathbb{R}^{n\times m} \to (\mathbb{R}^+)^m$. The function Q^{nn} takes as input a vector $\boldsymbol{x} \in \mathbb{R}^n$ which acts as input to the neural network and a matrix $\mathbf{T} \in \mathbb{R}^{n\times m}$ describing the weights of a single layer fully connected neural network. It computes the output of neural network inferencing as $\mathsf{Q}^{\mathrm{nn}}(\boldsymbol{x}, \mathbf{T}) = \mathrm{ReLU}^*(\boldsymbol{x} \times \mathbf{T})$. Here, the neural network uses the Rectified Linear Units (ReLU) [2] function, $\mathrm{ReLU}\colon \mathbb{R} \to \mathbb{R}^+$, which on input $x \in \mathbb{R}$ is evaluated as $\mathrm{ReLU}(x) = \max(0, x)$ to provide non-linearity. By $\mathrm{ReLU}^*\colon \mathbb{R}^n \to (\mathbb{R}^+)^n$ we denote the function which given a vector performs element-wise ReLU operation to generate the output vector. The two-party functionality for secure inferencing of a single layer fully connected neural network is given as $\mathsf{F}^{\mathrm{nn}}\colon \mathbb{R}^n \times \mathbb{R}^{n\times m} \to (\mathbb{R}^+)^m \times \{\varepsilon\}$ which on inputs $\boldsymbol{x}$ from party 1 (client) and $\mathbf{T}$ from party 2 (server) is evaluated as $\mathsf{F}^{\mathrm{nn}}(\boldsymbol{x}, \mathbf{T}) = (\mathsf{Q}^{\mathrm{nn}}(\boldsymbol{x}, \mathbf{T}), \varepsilon)$.

Inverters for F^{nn}. The inverter $\mathrm{IA}^{\mathrm{nn}}_h$ for F^{nn} functionality and honest party $h \in \{1, 2\}$ are shown in the right part of Figure 17.

For $h = 1$, the inverter $\mathrm{IA}^{\mathrm{nn}}_1$ just has to return any vector $\boldsymbol{x} \in \mathbb{R}^n$ as the output of party 2 is always ε. It gets the dimension n from its input $\mathbf{T} \in \mathbb{R}^{n\times m}$ which is the input of party 2.

For $h = 2$, the inverter $\mathrm{IA}^{\mathrm{nn}}_2$ gets as inputs the input of party 1 to F^{nn} which is a vector $\boldsymbol{x} \in \mathbb{R}^n$ and the output for party 1 which is a vector $\boldsymbol{y} \in \mathbb{R}^n$ such that for some matrix $\mathbf{T} \in \mathbb{R}^{n\times m}$ (unknown to party 1), $\boldsymbol{y} = \mathsf{F}^{\mathrm{nn}}(\boldsymbol{x}, \mathbf{T})[1]$. The inverter needs to output a matrix $\mathbf{T}'$ such that $\mathsf{F}^{\mathrm{nn}}(\boldsymbol{x}, \mathbf{T}')[1] = \boldsymbol{y}$. It first gets n and m from $\boldsymbol{x}, \boldsymbol{y}$ and then constructs a diagonal matrix $\mathbf{T}'$ such that $\boldsymbol{x} \times \mathbf{T}' = \boldsymbol{y}$ which makes it a correct inverter.

While the inverters seem efficient, one must be wary of working with elements in $\mathbb{R}$ and an appropriate finite field must be chosen so that the elements can be represented efficiently.

F Proof of Theorem 3.5

Proof of Theorem 3.5: Let b be the bit initialized in $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F},\Pi,h}$ and $q = \mathrm{Q}^{\mathrm{Run}}(A_{\mathrm{ini}})$. We will use hybrids $H^0_{\mathsf{F},\Pi,h}, \ldots, H^q_{\mathsf{F},\Pi,h}$ where $H^i_{\mathsf{F},\Pi,h}$ is as shown in Figure 18. In $H^i_{\mathsf{F},\Pi,h}$, the first i queries to the $\mathrm{Run}(x_0, x_1, x)$ oracle are answered with the transcript of executing the protocol with honest party input as x_0. The rest use x_1 as the input for honest party. The adversarial party input is always set to x. Thus, for adversary A_{ini} which makes q queries to the Run oracle of $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F},\Pi,h}$ the

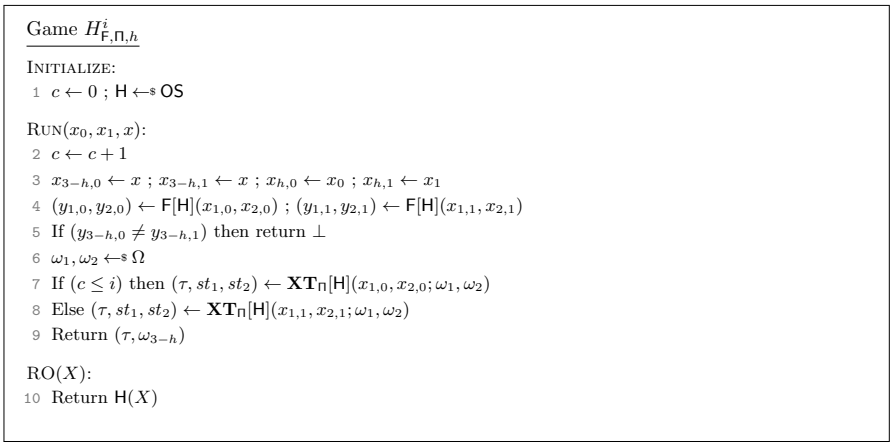


Figure 18: Hybrid $H^i_{\mathsf{F},\Pi,h}$ for the hybrid argument used to proof Theorem 3.5

hybrid $H^0_{\mathsf{F},\Pi,h}$ behaves the same as $\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,h}$ with $b = 0$ and $H^q_{\mathsf{F},\Pi,h}$ behaves as $\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,h}$ with $b = 1$. Let b' be the output of A_{ini} when playing the game $\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,h}$. This gives us

$$\Pr\left[\, b' = 1 \mid b = 1 \,\right] = \Pr[H^q_{\mathsf{F},\Pi,h}(A_{\text{ini}})]\ ,$$
$$\Pr\left[\, b' = 1 \mid b = 0 \,\right] = \Pr[H^0_{\mathsf{F},\Pi,h}(A_{\text{ini}})]\ .$$

Adversary B_{ini} is playing the game $\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,h}$ and can only make a single query to Run oracle. It starts by selecting an integer $i \leftarrow\!\!\$\ \{1, \ldots, q\}$ and initializing $c \leftarrow 0$. It then runs A_{ini} which is also playing the game $\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,h}$ and which makes q queries to its Run oracle. When A_{ini} makes a query Run(x_0, x_1, x), B_{ini} does the following:

1. $c \leftarrow c + 1$
2. $x_{3-h,0} \leftarrow x$; $x_{3-h,1} \leftarrow x$; $x_{h,0} \leftarrow x_0$; $x_{h,1} \leftarrow x_1$
3. $(y_{1,0}, y_{2,0}) \leftarrow \mathsf{F}[\text{RO}](x_{1,0}, x_{2,0})$
4. $(y_{1,1}, y_{2,1}) \leftarrow \mathsf{F}[\text{RO}](x_{1,1}, x_{2,1})$
5. If $(y_{3-h,0} \neq y_{3-h,1})$ then return $\perp$
6. $\omega_1, \omega_2 \leftarrow\!\!\$\ \Omega$
7. If $(c < i)$ then $(\tau, st_1, st_2) \leftarrow \mathbf{XT}_\Pi[\text{RO}](x_{1,0}, x_{2,0}; \omega_1, \omega_2)$
8. Else If $(c = i)$ then $(\tau, \omega_{3-h}) \leftarrow$ Run(x_0, x_1, x)
9. Else $(\tau, st_1, st_2) \leftarrow \mathbf{XT}_\Pi[\text{RO}](x_{1,1}, x_{2,1}; \omega_1, \omega_2)$
10. Return (τ, ω_{3-h})

Here, RO, Run oracles are the $\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,h}$.RO and $\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,h}$.Run oracles respectively that B_{ini} has access to in its game. In the above, whenever $(c < i)$, x_0 is used as honest party input to generate the output of Run oracle query by executing the protocol. And whenever $(c > i)$ the x_1 is used as honest party input. When $(c = i)$, adversary B_{ini} queries its $\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,h}$.Run oracle to answer the query. When A_{ini} queries the RO(X) oracle, adversary B_{ini} forwards it to its own RO(X). Finally, B_{ini} returns the bit returned by A_{ini}. Let b' be the bit initialized in the $\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,h}$ game. From the above we can see that when $b' = 1$, B_{ini} simulates the hybrid $H^i_{\mathsf{F},\Pi,h}$ for some $1 \leq i \leq q$. And in the other case when $b' = 0$, B_{ini} simulates the hybrid $H^{i-1}_{\mathsf{F},\Pi,h}$. Let b be the bit returned by B_{ini}. Since B_{ini}

randomly selects i, we have

$$\Pr\left[\, b = 1 \mid b' = 1 \,\right] = \frac{1}{q} \cdot \sum_{i=1}^{q} \Pr[H^{i}_{\mathsf{F},\Pi,h}(A_{\text{ini}})] \;,$$

$$\Pr\left[\, b = 1 \mid b' = 0 \,\right] = \frac{1}{q} \cdot \sum_{i=1}^{q} \Pr[H^{i-1}_{\mathsf{F},\Pi,h}(A_{\text{ini}})] \;.$$

This gives us

$$\begin{aligned}
\mathbf{Adv}^{\text{ini}}_{\mathsf{F},\Pi,h}(B_{\text{ini}}) &= \Pr\left[\, b = 1 \mid b' = 1 \,\right] - \Pr\left[\, b = 1 \mid b' = 0 \,\right] \\
&= \frac{1}{q} \left(\sum_{i=1}^{q} \Pr[H^{i}_{\mathsf{F},\Pi,h}(A_{\text{ini}})] - \sum_{i=1}^{q} \Pr[H^{i-1}_{\mathsf{F},\Pi,h}(A_{\text{ini}})] \right) \\
&= \frac{1}{q} \left(\Pr[H^{q}_{\mathsf{F},\Pi,h}(A_{\text{ini}})] - \Pr[H^{0}_{\mathsf{F},\Pi,h}(A_{\text{ini}})] \right) \;,
\end{aligned}$$

which is

$$\mathbf{Adv}^{\text{ini}}_{\mathsf{F},\Pi,h}(B_{\text{ini}}) = \frac{1}{q} \cdot \mathbf{Adv}^{\text{ini}}_{\mathsf{F},\Pi,h}(A_{\text{ini}}) \;,$$

$$\mathbf{Adv}^{\text{ini}}_{\mathsf{F},\Pi,h}(A_{\text{ini}}) = q \cdot \mathbf{Adv}^{\text{ini}}_{\mathsf{F},\Pi,h}(B_{\text{ini}}) \;,$$

which is Eq. (6).∎

G Proof of Theorem 4.2

Proof of Theorem 4.2: We start with the correctness of Π in realizing F. For that, we construct an adversary A_{prf} playing game $\mathbf{G}^{\text{prf}}_{\mathsf{Q}}$ using the given adversary A_{psi} playing game $\mathbf{G}^{\text{corr}}_{\mathsf{F},\Pi}$. Adversary A_{prf} initializes $i \leftarrow 0, \mathsf{win} \leftarrow 0$ and runs A_{psi}. It answers RO oracle queries from A_{psi} using its own RO oracle. To process a $\textsc{Run}(S_1, S_2)$ query from A_{psi}, adversary A_{rcr} does the following

1. $\textsc{New}$; $i \leftarrow i + 1$
2. For each $x \in S_1$ do
3. $\quad Z \leftarrow Z \cup \{\text{CH}(i, x)\}$
4. For each $x \in S_2$ do
5. $\quad$ If $\text{CH}(i, x) \in Y$ then $I \leftarrow I \cup \{x\}$
6. $\mathsf{win} \leftarrow [[I \neq (S_1 \cap S_2)]]$; Return win

Finally, when A_{psi} completes, A_{prf} returns the bit $b' \leftarrow [[\mathsf{win} = 1]]$. Let b be the challenge bit in the game $\mathbf{G}^{\text{prf}}_{\mathsf{Q}}$.

As mentioned before, for simplicity, our definition requires perfect correctness of Π^{oprf}. And so, if CH oracle query outputs are real values ($b = 1$), adversary A_{prf} correctly simulates the $\textsc{Run}$ oracle for A_{psi}. We get $\Pr\left[\, b' = 1 \mid b = 0 \,\right] = \mathbf{Adv}^{\text{corr}}_{\mathsf{F},\Pi}(A_{\text{psi}})$.

When $b = 0$, the values for CH oracle query outputs are picked randomly from R. Therefore, the win flag is set only if across all $\textsc{Run}$ queries the same value is picked for some element in $S_1 \setminus (S_1 \cap S_2)$

and some element in S_2. This gives us

$$\Pr\left[\, b' = 1 \mid b = 0 \,\right] \leq \sum_{i=1}^{q} \frac{s_{i,1} s_{i,2}}{|\mathsf{R}|} \ .$$

Combining the above, we get

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{Q}}(A_{\mathrm{prf}}) \geq \mathbf{Adv}^{\mathrm{corr}}_{\mathsf{F},\Pi}(A_{\mathrm{psi}}) - \sum_{i=1}^{q} \frac{s_{i,1} s_{i,2}}{|\mathsf{R}|} \ ,$$

which is Eq. (8).

We now show client security for Π. Using the given adversary A_{psi} playing game $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F},\Pi,1}$ we construct an adversary A_{oprf} playing game $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F}^{\mathrm{oprf}}_{\mathsf{Q}},\Pi^{\mathrm{oprf}},1}$. In the following, RUN and RO denote the oracles available to A_{oprf} in its game. Adversary A_{oprf} starts by running A_{psi}. When A_{psi} queries $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F},\Pi,1}.\textsc{Run}(S_{1,0}, S_{1,1}, S_2)$, adversary A_{oprf} does the following

1. If $(\mathsf{F}(S_{1,0}, S_2)[2] \neq \mathsf{F}(S_{1,1}, S_2)[2])$ then return $\bot$
2. $k \leftarrow\!\!\$\ \mathsf{Keys}$; $\boldsymbol{x}_0 \leftarrow \mathsf{S2V}(S_{1,0})$; $\boldsymbol{x}_1 \leftarrow \mathsf{S2V}(S_{1,1})$
3. $(\tau, \omega) \leftarrow \textsc{Run}(\boldsymbol{x}_0, \boldsymbol{x}_1, k)$
4. $Z \leftarrow \emptyset$
5. For each $x \in S_2$ do
6. $\quad Z \leftarrow Z \cup \mathsf{Q}[\mathrm{RO}](k, x)$
8. Return $((\tau, \mathsf{S2V}(Z)), (k, \omega))$

When A_{psi} queries $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F},\Pi,1}.\mathrm{RO}(X)$, adversary A_{oprf} makes queries $y \leftarrow \mathrm{RO}(X)$ and responds with y. Finally, adversary A_{oprf} outputs the bit that A_{psi} outputs. In the above, A_{oprf} creates the transcript of execution of Π on the inputs provided by A_{psi}. Adversary A_{oprf} selects the key k for Q that is supposed to be selected by the server in Π. It then uses k and the inputs for the client provided by A_{psi} to make its RUN oracle query. This returns the transcript for the Π^{oprf} execution on one of $S_{1,0}, S_{1,1}$ determined by the challenge bit of $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F}^{\mathrm{oprf}}_{\mathsf{Q}},\Pi^{\mathrm{oprf}},1}$. Consequently, this also determines the challenge bit for A_{psi}. To complete the transcript for Π, adversary A_{oprf} constructs the set $Z = \{\, \mathsf{Q}[\mathrm{RO}](k, x) \,:\, x \in S_2 \,\}$. Since A_{psi} and A_{oprf} are trying to guess the same challenge bit, we have Eq. (9).

Next we look at server security for Π. We construct adversary A_{oprf} playing game $\mathbf{G}^{\text{oprf-pr}}_{\Pi^{\mathrm{oprf}},\mathsf{Q}}$ using the given adversary A_{psi} which is playing game $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F},\Pi,2}$. It starts by selecting a bit $c \leftarrow\!\!\$\ \{0,1\}$ and initializing $i \leftarrow 0$. It then runs A_{psi}. When A_{psi} makes a query $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F},\Pi,2}.\mathrm{RO}(X)$, adversary A_{oprf} makes the query $\mathbf{G}^{\text{oprf-pr}}_{\Pi^{\mathrm{oprf}},\mathsf{Q}}.\mathrm{RO}(X)$ and responds with the output. When A_{psi} makes a query $\textsc{Run}(S_{2,0}, S_{2,1}, S_1)$, adversary A_{oprf} does the following:

1. If $(\mathsf{F}(S_1, S_{2,0})[1] \neq \mathsf{F}(S_1, S_{2,1})[1])$ then return $\bot$
2. $\textsc{New}$; $i \leftarrow i + 1$; $\boldsymbol{x}_1 \leftarrow \mathsf{S2V}(S_1)$
3. $(\boldsymbol{y}, \tau, \omega) \leftarrow \mathrm{TR}(i, \boldsymbol{x}_1)$; $Z \leftarrow \emptyset$
4. For $j = 1, \ldots, |\boldsymbol{x}_1|$ do
5. $\quad$ If $\boldsymbol{x}_1[j] \in S_{2,c}$ then $Z \leftarrow Z \cup \{\boldsymbol{y}[j]\}$
6. For each $x \in S_{2,c} \setminus S_1$ do
7. $\quad y \leftarrow \mathrm{CH}(i, x)$; $Z \leftarrow Z \cup \{y\}$
8. Return $((\tau, \mathsf{S2V}(Z)), \omega)$

In the above, A_{oprf} is creating the transcript of running Π with $S_1, S_{2,c}$ as the inputs of the parties. It uses its NEW oracle to initialize a new key. For the elements in the client's input set S_1, it generates the transcript of executing Π^{oprf} using the TR oracle. This also provides the evaluations of $\mathsf{Q}[\text{RO}](k, \cdot)$ on elements in S_1 in the vector $\boldsymbol{y}$, where k represents the key generated inside NEW. It generates the set $Z = \{\, \mathsf{Q}[\text{RO}](k, x) \,:\, x \in S_{2,c} \,\}$ by using $\boldsymbol{y}$ for elements in $S_{2,c} \cap S_1$ and its CH oracle for other values in $S_{2,c}$.

Let c' be the output from A_{psi}. Finally, A_{oprf} lets $b' \leftarrow [[c = c']]$ and returns b' as its own guess for the challenge bit b selected in its own game, meaning game $\mathbf{G}^{\text{oprf-pr}}_{\Pi^{\text{oprf}},\mathsf{Q}}$. If the CH oracle returns the output of evaluating Q, then from the above we can see that A_{oprf} returns a correct transcript of executing Π with the inputs $S_1, S_{2,c}$ for the parties. Thus, we get

$$\Pr\left[\, b' = 1 \mid b = 1 \,\right] = \Pr[\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,2}(A_{\text{psi}})] \,.$$

On the other hand, if the CH oracle returns random values, A_{psi} is provided with a transcript in which the output of OPRF evaluations on $S_{2,c}$ are random values from R. Since this does not provide any information on c and all the other steps are independent of c, the adversary A_{psi} can only guess the bit c. This gives

$$\Pr\left[\, b' = 1 \mid b = 0 \,\right] = \frac{1}{2} \,.$$

From the above we get,

$$\begin{aligned}\mathbf{Adv}^{\text{oprf-pr}}_{\Pi^{\text{oprf}},\mathsf{Q}}(A_{\text{oprf}}) &= \Pr\left[\, b' = 1 \mid b = 1 \,\right] - \Pr\left[\, b' = 1 \mid b = 0 \,\right] \\ &= \Pr[\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,2}(A_{\text{psi}})] - \frac{1}{2} = \frac{1}{2} \cdot \mathbf{Adv}^{\text{ini}}_{\mathsf{F},\Pi,2}(A_{\text{psi}}) \,,\end{aligned}$$

which gives us Eq. (10). ∎

H Proof of Theorem 5.1

Proof of Theorem 5.1: We show that we can construct an adversary $A_{\text{v-cdh}}$ from A_{v} such that Eq. (11) holds. Let $q_{\text{nk}} = \text{Q}^{\text{NEWKEY}}(A_{\text{v}})$ and $q_{\text{nb}} = \text{Q}^{\text{NEWBASE}}(A_{\text{v}})$. Adversary $A_{\text{v-cdh}}$ which is playing the $\mathbf{G}^{\text{v-cdh}}_{\mathbb{G},g,p}$ game uses A_{v} which is playing the $\mathbf{G}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}$ game. $A_{\text{v-cdh}}$ gets as input a challenge (K^*, B^*). First $A_{\text{v-cdh}}$ selects an integer i^* uniformly at random from $\{1, \ldots, q_{\text{nk}}\}$ and an integer j^* uniformly at random from $\{1, \ldots, q_{\text{nb}}\}$. It also sets $i \leftarrow 0$ and $j \leftarrow 0$. When A_{v} makes a query to its NEWKEY oracle, $A_{\text{v-cdh}}$ does the following:

1. $i \leftarrow i + 1$
2. If $(i = i^*)$ then return K^*
3. $k_i \leftarrow\!\!\$\ \mathbb{Z}_p$; $K_i \leftarrow g^{k_i}$
4. Return K_i

When A_{v} makes a query to NEWBASE, $A_{\text{v-cdh}}$ proceeds similar, incrementing j and picking $b_j \leftarrow\!\!\$\ \mathbb{Z}_p$ to return $B_j \leftarrow g^{b_j}$ or B^* if $j = j^*$. When $A_{\text{v-cdh}}$ makes a query to the CDHO(i', j'), then $A_{\text{v-cdh}}$ does the following:

1. If not $(i' \leq i)$ or not $(j' \leq j)$ then output $\perp$
2. If $(i' = i^*)$ and $(j' = j^*)$ then abort

3. If $(i' \neq i^*)$ then return $B_{j'}^{k_{i'}}$
4. Return $K_{i'}^{b_{j'}}$

Note that $A_{\text{v-cdh}}$ can reply to all queries except when $i' = i^*$ and $j' = j^*$ in which case it will abort. When A_{v} queries the $\mathbf{G}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}.\text{DDHO}(i', j', Z)$ and $i' = i^*$ and $j' = j^*$, $A_{\text{v-cdh}}$ queries its own oracle $\mathbf{G}^{\text{v-cdh}}_{\mathbb{G},g,p}.\text{DDHO}(Z)$ and forwards the response. Otherwise it can use one of the secrets $k_{i'}$ or $b_{j'}$ as above.

Finally, A_{v} outputs (i', j', Z') as its solution. Let E be the event that $i' = i^*$ and $j' = j^*$, in which case $A_{\text{v-cdh}}$ stops with output Z'. Otherwise, it aborts. Then we have

$$\begin{aligned}\Pr[\mathbf{G}^{\text{v-cdh}}_{\mathbb{G},g,p}(A_{\text{v-cdh}})] &= \Pr[\mathbf{G}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}(A_{\text{v}}) \cap E] \\ &= \Pr[\mathbf{G}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}(A_{\text{v}})] \cdot \Pr[E] \\ &\geq \Pr[\mathbf{G}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}(A_{\text{v}})] \cdot \frac{1}{q_{\text{nk}} q_{\text{nb}}} \ .\end{aligned}$$

This gives us

$$\mathbf{Adv}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}(A_{\text{v}}) \leq q_{\text{nk}} q_{\text{nb}} \cdot \mathbf{Adv}^{\text{v-cdh}}_{\mathbb{G},g,p}(A_{\text{v-cdh}}) \ ,$$

which is Eq. (11). Observe that we can prove Eq. (12) in the same way, omitting the DDHO.

To bound the running time, note that $A_{\text{v-cdh}}$ performs one exponentiation for each of A_{v}'s oracle queries, unless the query involves i^* or j^*. ∎

I Proof of Theorem 5.2

Proof of Theorem 5.2: We use the fact that DDH ⇒ DDH-MU with a tight reduction by a standard re-randomization argument. We construct an adversary $A_{\text{ddh-mu}}$ that plays in game $\mathbf{G}^{\text{ddh-mu}}_{\mathbb{G},g,p}$ and uses an adversary A_{v} playing in game $\mathbf{G}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}$. Adversary $A_{\text{ddh-mu}}$ responds to oracle queries as follows. When A_{v} queries $\mathbf{G}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}.\textsc{NewKey}$ or $\mathbf{G}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}.\textsc{NewBase}$, $A_{\text{ddh-mu}}$ simply calls its own oracles and forwards the group elements K_i and B_j, respectively. When A_{v} queries its $\text{CDHO}(i', j')$ oracle, $A_{\text{ddh-mu}}$ queries $\text{CH}(i', j')$ to get a DDH challenge $Z_{i',j'}$ and gives it to A_{v}. Similarly, when A_{v} queries its $\text{DDHO}(i', j', Z')$ and $A_{\text{ddh-mu}}$ has not yet asked for $Z_{i',j'}$, it does this first and then replies with $[[Z_{i',j'} = Z']]$. Finally, A_{v} will output a solution (i', j', Z'). Again, if $A_{\text{ddh-mu}}$ has not asked its CH oracle yet, it will do so. Then if $[[Z_{i',j'} = Z']]$, $A_{\text{ddh-mu}}$ stops with 1 ("real"). Otherwise, it stops with 0 ("random").

Let c be the challenge bit used in game $\mathbf{G}^{\text{ddh-mu}}_{\mathbb{G},g,p}$. To analyze the advantage of $A_{\text{ddh-mu}}$, we observe that if $c = 1$, then $A_{\text{ddh-mu}}$ perfectly simulates $\mathbf{G}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}$ for A_{v} and it wins whenever A_{v} wins. We get

$$\Pr[A_{\text{ddh-mu}} \Rightarrow 1 \mid c = 1] = \Pr[\mathbf{G}^{\text{cdh-muc}}_{\mathbb{G},g,p}(A_{\text{v}})] \ .$$

If $c = 0$, then $A_{\text{ddh-mu}}$ outputs random group elements to A_{v} and also uses those to answer DDHO queries. We analyze the probability that $A_{\text{ddh-mu}}$ nevertheless outputs 1. This will happen if A_{v} queries the DDHO oracle on one of the random group elements $Z_{i',j'}$ for any pair (i', j') for which it has not previously queried the CDHO. Due to the latter, such a $Z_{i',j'}$ is independent of A_{v}'s view. Thus, for each query, the probability is $1/p$. If no such query happens, this may for the final

output of A_{v}. Therefore, we get

$$\Pr[A_{\text{ddh-mu}} \Rightarrow 1 \mid c = 0] \leq \frac{\mathrm{Q}^{\mathrm{DDHO}}(A_{\mathrm{v}}) + 1}{p} ,$$

Putting the above together gives us

$$\begin{aligned}\mathbf{Adv}^{\text{ddh-mu}}_{\mathbb{G},g,p}(A_{\text{ddh-mu}}) &= \Pr[A_{\text{ddh-mu}} \Rightarrow 1 \mid c = 1] - \Pr[A_{\text{ddh-mu}} \Rightarrow 1 \mid c = 0] \\ &\geq \Pr[\mathbf{G}^{\text{cdh-muc}}_{\mathbb{G},g,p}(A_{\mathrm{v}})] - \frac{\mathrm{Q}^{\mathrm{DDHO}}(A_{\mathrm{v}}) + 1}{p} ,\end{aligned}$$

which is Eq. (13) in Theorem 5.2. ∎

J Proof of Theorem 6.1

Proof of Theorem 6.1: Each RUN query of A_{ini} consists of a pair $\boldsymbol{x}_0, \boldsymbol{x}_1$ of equal-length vectors over $\{0,1\}^*$, and a key $k \in \mathbb{Z}_p$ for 2HDH. Let $\boldsymbol{x}_{0,i}, \boldsymbol{x}_{1,i}$ be the vectors in the i-th query. Let S be the union, over all $c \in \{0,1\}$ and $i \in \{1, \ldots, \mathrm{Q}^{\mathrm{RUN}}(A_{\text{ini}})\}$, of the sets $\mathsf{V2S}(\boldsymbol{x}_{c,i})$. Let BD be the event that there is an $x \in S$ with $\mathrm{RO}(1,x) = 1$ being the identity element of the group $\mathbb{G}$. If this bad event does not happen, then $w = \mathrm{RO}(1,x) \in \mathbb{G}^*$ is a generator for all $x \in S$, and thus w^r is uniformly distributed over $\mathbb{G}^*$ when $r \leftarrow\!\!\$\, \mathbb{Z}_p^*$. So if the bad event does not happen, the adversary has zero advantage. Its advantage is thus bounded above by the probability of BD. This is at most the probability that the execution of the game with A_{ini} makes a $\mathrm{RO}(1,\cdot)$ query that returns 1. Recalling that $\mathrm{Q}^{\mathrm{RO}}(A_{\text{ini}})$ counts both the queries made directly by A_{ini} and the ones made by the protocol in the execution of the game with A_{ini}, the probability of BD is at most $\mathrm{Q}^{\mathrm{RO}}(A_{\text{ini}})$. ∎

K Proof of Theorem 6.2

Proof of Theorem 6.2: We prove OPRF-PR security of 2HDH with different bounds, depending on which problem we use. We will first give a tight reduction from V-CDH-MUC. After that we show how to modify it for CDH-MUC, which incurs a loss linear in the number of random oracle queries. The bounds for V-CDH, CDH and DDH then follow from the relations proven in Theorems 5.1 and 5.2.

PROOF FOR XX = V-CDH-MUC. We will use the games G_0, G_1 and G_2 shown in Figure 19, where the boxed code is only included in G_0 and line 32 is only included in G_0 and G_1. In all games, the RO oracle runs the method H_1 when queried with signature $(1,\cdot)$ and H_2 when queried with signature $(2,\cdot,\cdot,\cdot)$. The H_1 method returns a random element from $\mathbb{G}$ and H_2 returns a random element from $\{0,1\}^\ell$. This is the same as what the random oracle for 2HDH does. In all games the TR is the same. When queried on input $(i', \boldsymbol{x})$, it generates the transcript $\tau = (\boldsymbol{b}, \boldsymbol{z}, g^{k_{i'}})$ with $\boldsymbol{b}[j'] = \mathsf{H}_1(\boldsymbol{x}[j'])^{\boldsymbol{r}[j']}$, $\boldsymbol{z}[j'] = \mathsf{H}_1(\boldsymbol{x}[j'])^{k_{i'}\boldsymbol{r}[j']}$ for key $k_{i'}$, $j' \in \{1, \ldots, |\boldsymbol{x}|\}$ and $\boldsymbol{r}[j'] \leftarrow\!\!\$\, \mathbb{Z}_p^*$. It generates the output $\boldsymbol{y}$ with $\boldsymbol{y}[j'] = \mathsf{H}_2(g^{k_{i'}}, \boldsymbol{x}[j'], \mathsf{H}_1(\boldsymbol{x}[j'])^{k_{i'}})$, and returns $(\boldsymbol{y}, \tau, \boldsymbol{r})$. This is same as TR oracle in $\mathbf{G}^{\text{oprf-pr}}_{\Pi_{\mathsf{2HDH}},\mathsf{2HDH}}$. The game further uses a set C_K to store keys queried to H_2 and sets a flag bad_1 if the NEW oracle picks a key that was previously queried to H_2. If bad_1 is set, then the CH will compute the output for $c = 1$ honestly using H_2. Otherwise, it will pick a fresh random value which it stores in a table T. If H_2 is later queried on the respective input, we set a flag bad_2 and return the random value stored in T. Therefore, G_0 proceeds exactly as $\mathbf{G}^{\text{oprf-pr}}_{\Pi_{\mathsf{2HDH}},\mathsf{2HDH}}$ and we

have

$$\Pr[\mathbf{G}^{\text{oprf-pr}}_{\Pi_{\mathsf{2HDH}},\mathsf{2HDH}}(A_{\text{oprf}})] = \Pr[\mathrm{G}_0(A_{\text{oprf}})] \ . \tag{22}$$

In G_1 we drop the boxed code, but we keep line 32 for now. Essentially, this step ensures that the final reduction will work and that the adversary does not query the random oracle before the key is picked. Games $\mathrm{G}_0, \mathrm{G}_1$ are identical-until-bad_1, so by the Fundamental Lemma of Game Playing [15] we have

$$\Pr[\mathrm{G}_0(A_{\text{oprf}})] = \Pr[\mathrm{G}_1(A_{\text{oprf}})] + \Pr[\mathrm{G}_0(A_{\text{oprf}})] - \Pr[\mathrm{G}_1(A_{\text{oprf}})] \tag{23}$$

$$\leq \Pr[\mathrm{G}_1(A_{\text{oprf}})] + \Pr[\mathrm{G}_1(A_{\text{oprf}}) \text{ sets } \mathsf{bad}_1] \ . \tag{24}$$

Since the keys are chosen uniformly at random from $\mathbb{Z}_p$, we can show that the probability that bad_1 is set is low. More specifically, since there are $\mathrm{Q}^{\text{NEW}}(A_{\text{oprf}})$ keys and $\mathrm{Q}^{\text{RO}}(A_{\text{oprf}})$ random oracle queries, we can upper bound the probability by

$$\Pr[\mathrm{G}_1(A_{\text{oprf}}) \text{ sets } \mathsf{bad}_1] \leq \frac{\mathrm{Q}^{\text{NEW}}(A_{\text{oprf}}) \cdot \mathrm{Q}^{\text{RO}}(A_{\text{oprf}})}{p} \tag{25}$$

Finally, in 2 we also drop line 32. Since G_1 and G_2 only differ in the code after flag bad_2 is set, the Fundamental Lemma of Game Playing [15] gives us

$$\Pr[\mathrm{G}_1(A_{\text{oprf}})] = \Pr[\mathrm{G}_2(A_{\text{oprf}})] + \Pr[\mathrm{G}_1(A_{\text{oprf}})] - \Pr[\mathrm{G}_2(A_{\text{oprf}})] \tag{26}$$

$$\leq \Pr[\mathrm{G}_2(A_{\text{oprf}})] + \Pr[\mathrm{G}_2(A_{\text{oprf}}) \text{ sets } \mathsf{bad}_2] \ . \tag{27}$$

First, we argue that

$$\Pr[\mathrm{G}_2(A_{\text{oprf}})] = \frac{1}{2} \ , \tag{28}$$

which can be justified by observing that the adversary's view is independent of the challenge bit c. This is because CH outputs uniformly random values from $\{0,1\}^\ell$ independent of c and the table T is not used in H_2.

Finally, we bound the probability of bad_2 being set. For this we give an adversary $A_{\text{v-cdh-muc}}$ playing the $\mathbf{G}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}$ game using the adversary A_{oprf} which is playing game G_2. We show $A_{\text{v-cdh-muc}}$ on the left of Figure 20. Whenever A_{oprf} sets the bad_2 flag in G_2, $A_{\text{v-cdh-muc}}$ wins the $\mathbf{G}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}$ game. $A_{\text{v-cdh-muc}}$ uses an additional table T_{CH} to map input strings x to indices j'. On the j^{th} query to $\mathrm{RO}(1,\cdot)$, it queries its own CH and assigns j to $\mathrm{T}_{\text{CH}}[x]$. This is necessary to call oracles CDHO, DDHO and FINALIZE on the correct inputs. We get

$$\Pr[\mathrm{G}_2(A_{\text{oprf}}) \text{ sets } \mathsf{bad}_2] \leq \mathbf{Adv}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}(A_{\text{v-cdh-muc}}) \ , \tag{29}$$

Note that

$$\begin{aligned}
\mathrm{Q}^{\text{NEWKEY}}(A_{\text{v-cdh-muc}}) &= \mathrm{Q}^{\text{NEW}}(A_{\text{oprf}}) \\
\mathrm{Q}^{\text{NEWBASE}}(A_{\text{v-cdh-muc}}) &\leq \mathrm{Q}^{\text{RO}}(A_{\text{oprf}}) \\
\mathrm{Q}^{\text{DDHO}}(A_{\text{v-cdh-muc}}) &\leq \mathrm{Q}^{\text{RO}}(A_{\text{oprf}}) \\
\mathrm{Q}^{\text{CDHO}}(A_{\text{v-cdh-muc}}) &\leq |\boldsymbol{x}_1| + \cdots |\boldsymbol{x}_{q_{\text{tr}}}| \ ,
\end{aligned}$$

where $q_{\text{tr}} = \mathrm{Q}^{\text{TR}}(A_{\text{oprf}})$ and $|\boldsymbol{x}_j|$ is the length of the vector from the j-th TR query.

We now put the above equations together to conclude. For brevity, we let $g_i = \Pr[\mathrm{G}_i(A_{\text{oprf}})]$;

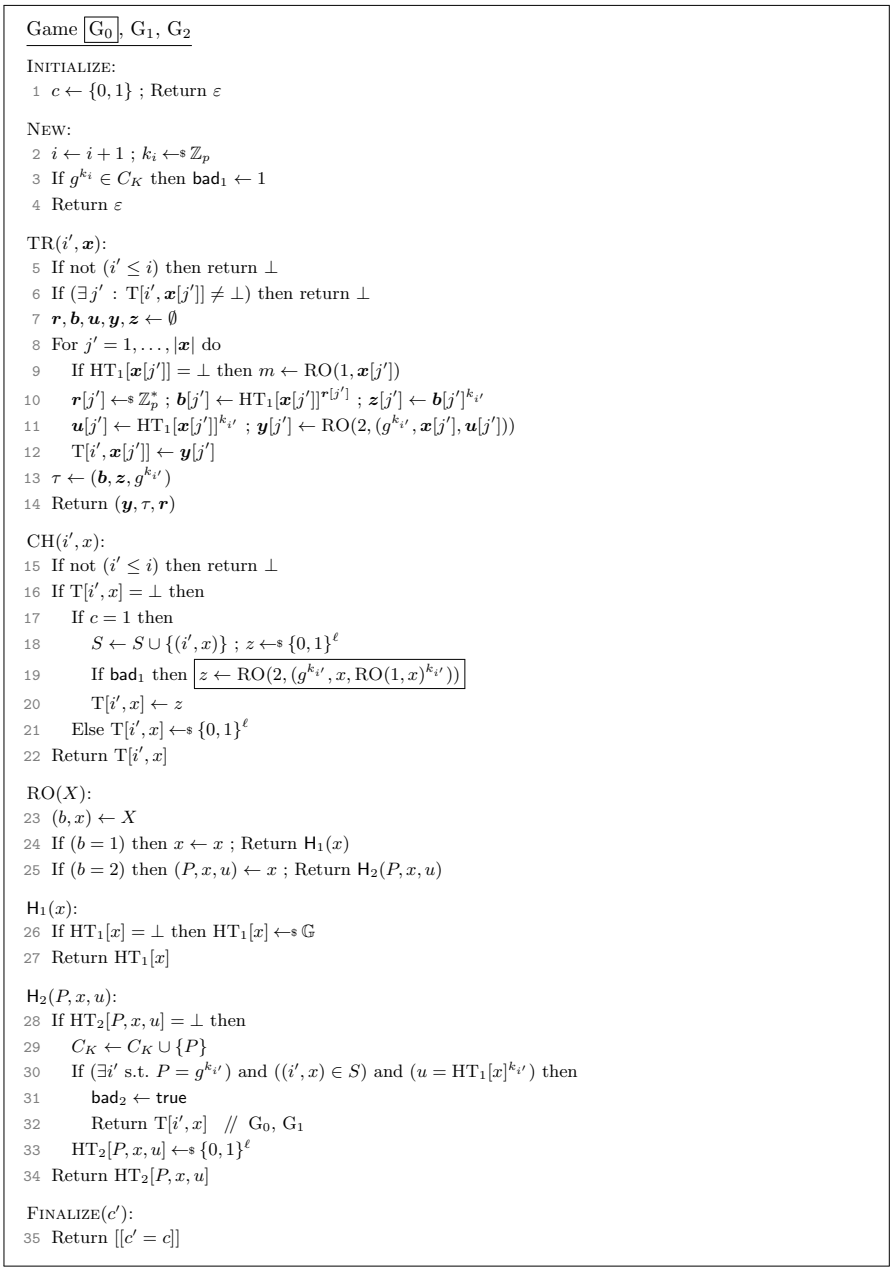


Figure 19: Games used for the analysis in proof of Theorem 6.2. The boxed code is only included in G_0. Line 32 is only included in G_0 and G_1.

$\epsilon = \mathbf{Adv}^{\text{oprf-pr}}_{\Pi_{\mathsf{2HDH}},\mathsf{2HDH}}(A_{\text{oprf}})$; $\epsilon' = \mathbf{Adv}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}(A_{\text{v-cdh-muc}})$; $\delta = (\mathrm{Q}^{\mathrm{NEW}}(A_{\text{oprf}}) \cdot \mathrm{Q}^{\mathrm{RO}}(A_{\text{oprf}}))/p$. Then from Eq. (22) through Eq. (29), we get

$$\begin{aligned}\epsilon = 2g_0 - 1 &= 2(g_0 - g_1) + (2g_1 - 1) \le 2\delta + (2g_1 - 1) \\ &= 2\delta + 2(g_1 - g_2) + (2g_2 - 1) = 2\delta + 2\epsilon' + (2(1/2) - 1) = 2\delta + 2\epsilon'\end{aligned}$$

which is the bound for xx = v-cdh-muc.

PROOF FOR xx = cdh-muc. We use the same games G_0, G_1 and G_2 described in Figure 19 and

used in the previous analysis. From combing Eq. (22) through Eq. (28), we get

$$\mathbf{Adv}^{\text{oprf-pr}}_{\Pi_{\mathsf{2HDH}},\mathsf{2HDH}}(A_{\text{oprf}}) \le 2\Pr[\mathrm{G}_2(A_{\text{oprf}}) \text{ sets } \mathsf{bad}_2] + \frac{2\mathrm{Q}^{\text{New}}(A_{\text{oprf}}) \cdot \mathrm{Q}^{\text{RO}}(A_{\text{oprf}})}{p} .$$

Here we want to bound bad_2 differently, namely via an adversary $A_{\text{cdh-muc}}$ playing the $\mathbf{G}^{\text{cdh-muc}}_{\mathbb{G},g,p}$ game. Let $q = \mathrm{Q}^{\text{RO}}(A_{\text{oprf}})$. On the right of Figure 20 we give $A_{\text{cdh-muc}}$ which uses adversary A_{oprf} playing the game $\mathbf{G}^{\text{oprf-pr}}_{\Pi^{\mathsf{2HDH}},\mathsf{2HDH}}$. On INITIALIZE, adversary $A_{\text{cdh-muc}}$ selects $j^* \leftarrow\!\!\$\ \{1,\ldots,q\}$ and initializes $t \leftarrow 0$. It simulates the TR and CH oracles as well as the H_1 method as the adversary $A_{\text{v-cdh-muc}}$ constructed in Theorem 6.2. The value of t is incremented by 1 in every RO query. When RO oracle is queried with input $(2, P, x, u)$ and the H_2 method is called, then the simulation differs from the previous proof because $A_{\text{cdh-muc}}$ does not have a DDHO oracle. Instead, when $(t = j^*)$ and P corresponds to a key $K_{i'}$ output by $A_{\text{cdh-muc}}$'s NEWKEY oracle and $\mathrm{T}_{\mathrm{CH}}[x] \neq \perp$, then H_2 calls FINALIZE for $(i', \mathrm{T}_{\mathrm{CH}}[x], u)$. Recall that if $\mathrm{RO}(1, x)$ has been queried, then $\mathrm{T}_{\mathrm{CH}}[x]$ stores some index j' for which $A_{\text{cdh-muc}}$ hopes to solve CDH. Otherwise it returns $z \leftarrow\!\!\$\ \{0,1\}^\ell$.

Let E be the event that the j^{*th} RO query that A_{oprf} makes is indeed of the form $(g^{k_{i'}}, x, \mathrm{HT}_1[x]^{k_{i'}})$ for some $i' \le i$ and x previously used in a CH query. We get

$$\Pr[\mathrm{G}_2(A_{\text{oprf}}) \text{ sets } \mathsf{bad}_2 \cap E] \le \mathbf{Adv}^{\text{cdh-muc}}_{\mathbb{G},g,p}(A_{\text{cdh-muc}}) .$$

By using that $\Pr[\mathrm{G}_2(A_{\text{oprf}}) \text{ sets } \mathsf{bad}_2 \cap E] = \Pr[\, E \mid \mathrm{G}_2(A_{\text{oprf}}) \text{ sets } \mathsf{bad}_2 \,] \cdot \Pr[\mathrm{G}_2(A_{\text{oprf}}) \text{ sets } \mathsf{bad}_2]$ and $\Pr[\, E \mid \mathrm{G}_2(A_{\text{oprf}}) \text{ sets } \mathsf{bad}_2 \,] \ge 1/q$, we get

$$\Pr[\mathrm{G}_2(A_{\text{oprf}}) \text{ sets } \mathsf{bad}_2] \le q \cdot \mathbf{Adv}^{\text{cdh-muc}}_{\mathbb{G},g,p}(A_{\text{cdh-muc}}) ,$$

which yields the bound for xx = cdh-muc. Finally, note that

$$\begin{aligned} \mathrm{Q}^{\text{NewKey}}(A_{\text{cdh-muc}}) &= \mathrm{Q}^{\text{New}}(A_{\text{oprf}}) \\ \mathrm{Q}^{\text{NewBase}}(A_{\text{cdh-muc}}) &\le \mathrm{Q}^{\text{RO}}(A_{\text{oprf}}) \\ \mathrm{Q}^{\text{CDHO}}(A_{\text{cdh-muc}}) &\le |\boldsymbol{x}_1| + \cdots |\boldsymbol{x}_{q_{\text{tr}}}| , \end{aligned}$$

where $q_{\text{tr}} = \mathrm{Q}^{\text{TR}}(A_{\text{oprf}})$ and $|\boldsymbol{x}_j|$ is the length of the vector from the j-th TR query.

PROOF FOR XX = V-CDH. We use the result from xx = v-cdh-muc and combine it with Eq. (11) from Theorem 5.1. The bound follows by observing that the number of NEWKEY and NEWBASE queries translate to the number of NEW and RO queries, respectively.

PROOF FOR XX = CDH. Here we use the result from xx = cdh-muc and combine it with Eq. (12) from Theorem 5.1. Similar to the previous case, the bound follows by observing that the number of NEWKEY and NEWBASE queries translate to the number of NEW and RO queries, respectively.

PROOF FOR XX = DDH. We use the result from xx = v-cdh-muc and combine it with Theorem 5.2. This gives a tight bound based on the DDH problem assuming that the additional term is negligible, which is the case for standard group sizes such as $\log(p) = 256$.

This concludes the proof of Theorem 6.2. ∎

Adversary $A_{\text{v-cdh-muc}}$

▷ Run A_{oprf}, responding to its oracle queries as follows:

NEW:
$i \leftarrow i+1$; $K_i \leftarrow \mathbf{G}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}.\text{NEWKEY}$; Return ε

TR$(i', \boldsymbol{x})$:
If not $(i' \le i)$ then return $\perp$
If $(\exists j' : \text{T}[i', \boldsymbol{x}[j']] \neq \perp)$ then return $\perp$
$\boldsymbol{r}, \boldsymbol{b}, \boldsymbol{u}, \boldsymbol{y}, \boldsymbol{z} \leftarrow \emptyset$
For $j' = 1, \ldots, |\boldsymbol{x}|$ do
  If $\text{HT}_1[\boldsymbol{x}[j']] = \perp$ then $m \leftarrow \text{RO}(1, \boldsymbol{x}[j'])$
  $\boldsymbol{r}[j'] \leftarrow\!\!\$\ \mathbb{Z}_p^*$
  $\boldsymbol{b}[j'] \leftarrow \text{HT}_1[\boldsymbol{x}[j']]^{\boldsymbol{r}[j']}$
  $\boldsymbol{z}[j'] \leftarrow \mathbf{G}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}.\text{CDHO}(i', \text{T}_{\text{CH}}[\boldsymbol{x}[j']])$
  $\boldsymbol{y}[j'] \leftarrow\!\!\$\ \{0,1\}^\ell$
  $\text{T}[i', \boldsymbol{x}[j']] \leftarrow \boldsymbol{y}[j']$
$\tau \leftarrow (\boldsymbol{b}, \boldsymbol{z}, K_{i'})$
Return $(\boldsymbol{y}, \tau, \boldsymbol{r})$

CH(i', x):
If not $(i' \le i)$ then return $\perp$
If $\text{T}[i', x] = \perp$ then
  $S \leftarrow S \cup \{(i', x)\}$
  $\text{T}[i', x] \leftarrow\!\!\$\ \{0,1\}^\ell$
Return $\text{T}[i', x]$

RO(X'):
$(b, x) \leftarrow X'$
If $(b = 1)$ then $x \leftarrow x$; Return $\mathsf{H}_1(x)$
If $(b = 2)$ then $(P, x, u) \leftarrow x$; Return $\mathsf{H}_2(P, x, u)$

$\mathsf{H}_1(x)$:
If $\text{HT}_1[x] = \perp$ then
  $j \leftarrow j+1$; $\text{T}_{\text{CH}}[x] \leftarrow j$
  $\text{HT}_1[x] \leftarrow \mathbf{G}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}.\text{NEWBASE}$
Return $\text{HT}_1[x]$

$\mathsf{H}_2(P, x, u)$:
If $\text{HT}_2[P, x, u] = \perp$ then
  If $(\exists i' \text{ s.t. } P = K_{i'})$ and $((i', x) \in S)$
  and $(\mathbf{G}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}.\text{DDHO}(i', \text{T}_{\text{CH}}[x], u))$ then
    $\mathbf{G}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}.\text{FINALIZE}(i', \text{T}_{\text{CH}}[x], u)$
  $\text{HT}_2[P, x, u] \leftarrow\!\!\$\ \{0,1\}^\ell$
Return $\text{HT}_2[P, x, u]$

Adversary $A_{\text{cdh-muc}}$

▷ Run A_{oprf}, responding to its oracle queries as follows:

INITIALIZE:
$j^* \leftarrow\!\!\$\ \{1, \ldots, q\}$; $t \leftarrow 0$; Return ε

NEW:
$i \leftarrow i+1$; $K_i \leftarrow \mathbf{G}^{\text{cdh-muc}}_{\mathbb{G},g,p}.\text{NEWKEY}$; Return ε

TR$(i', \boldsymbol{x})$:
If not $(i' \le i)$ then return $\perp$
If $(\exists j' : \text{T}[i', \boldsymbol{x}[j']] \neq \perp)$ then return $\perp$
$\boldsymbol{r}, \boldsymbol{b}, \boldsymbol{u}, \boldsymbol{y}, \boldsymbol{z} \leftarrow \emptyset$
For $j' = 1, \ldots, |\boldsymbol{x}|$ do
  If $\text{HT}_1[\boldsymbol{x}[j']] = \perp$ then $m \leftarrow \text{RO}(1, \boldsymbol{x}[j'])$
  $\boldsymbol{r}[j'] \leftarrow\!\!\$\ \mathbb{Z}_p^*$
  $\boldsymbol{b}[j'] \leftarrow \text{HT}_1[\boldsymbol{x}[j']]^{\boldsymbol{r}[j']}$
  $\boldsymbol{z}[j'] \leftarrow \mathbf{G}^{\text{cdh-muc}}_{\mathbb{G},g,p}.\text{CDHO}(i', \text{T}_{\text{CH}}[\boldsymbol{x}[j']])$
  $\boldsymbol{y}[j'] \leftarrow\!\!\$\ \{0,1\}^\ell$
  $\text{T}[i', \boldsymbol{x}[j']] \leftarrow \boldsymbol{y}[j']$
$\tau \leftarrow (\boldsymbol{b}, \boldsymbol{z}, K_{i'})$
Return $(\boldsymbol{y}, \tau, \boldsymbol{r})$

CH(i', x):
If not $(i' \le i)$ then return $\perp$
If $\text{T}[i', x] = \perp$ then
  $S \leftarrow S \cup \{(i', x)\}$
  $\text{T}[i', x] \leftarrow\!\!\$\ \{0,1\}^\ell$
Return $\text{T}[i', x]$

RO(X'):
$(b, x) \leftarrow X'$; $t \leftarrow t+1$
If $(b = 1)$ then $x \leftarrow x$; Return $\mathsf{H}_1(x)$
If $(b = 2)$ then $(P, x, u) \leftarrow x$; Return $\mathsf{H}_2(P, x, u)$

$\mathsf{H}_1(x)$:
If $\text{HT}_1[x] = \perp$ then
  $j \leftarrow j+1$; $\text{T}_{\text{CH}}[x] \leftarrow j$
  $\text{HT}_1[x] \leftarrow \mathbf{G}^{\text{cdh-muc}}_{\mathbb{G},g,p}.\text{NEWBASE}$
Return $\text{HT}_1[x]$

$\mathsf{H}_2(P, x, u)$:
If $\text{HT}_2[P, x, u] = \perp$ then
  If $(t = j^*)$ and $(\exists i' \text{ s.t. } P = g^{k_{i'}})$ then
    $\mathbf{G}^{\text{cdh-muc}}_{\mathbb{G},g,p}.\text{FINALIZE}(i', j^*, u)$
  $\text{HT}_2[P, x, u] \leftarrow\!\!\$\ \{0,1\}^\ell$
Return $\text{HT}_2[P, x, u]$

Figure 20: **Left:** Adversary $A_{\text{v-cdh-muc}}$ playing $\mathbf{G}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}$ using A_{oprf} which is playing the OPRF-PR game. **Right:** Adversary $A_{\text{cdh-muc}}$ playing $\mathbf{G}^{\text{cdh-muc}}_{\mathbb{G},g,q}$ using A_{oprf} which is playing the OPRF-PR game.

L Proof of Theorem 7.1

Proof of Theorem 7.1: Starting with correctness, a violation of it can only occur if there are i, j such that $\boldsymbol{x}_1[i] \neq \boldsymbol{x}_2[j]$ but $\mathsf{H}_2(\eta, K, \boldsymbol{x}_1[i], \boldsymbol{d}[i]) = \mathsf{H}_2(\eta, K, \boldsymbol{x}_2[j], \boldsymbol{c}[j])$, meaning there is a collision for H_2 for some $\eta \in \{0,1\}^{sl}$ and $K \in \mathbb{G}$. To bound the collision probability, suppose that the sizes of the two sets in the adversary's i-th RUN query are $s_{i,1}, s_{i,2}$, respectively. Then

$$\mathbf{Adv}^{\text{corr}}_{\mathsf{F},\Pi}(A_{\text{corr}}) \le \sum_i \frac{s_{i,1} \cdot s_{i,2}}{2^{hl}} \le \frac{M^2}{2^{hl}} \ ,$$

which is Eq. (17).

We turn to client security which is almost identical to that of 2HDH. The only difference is that

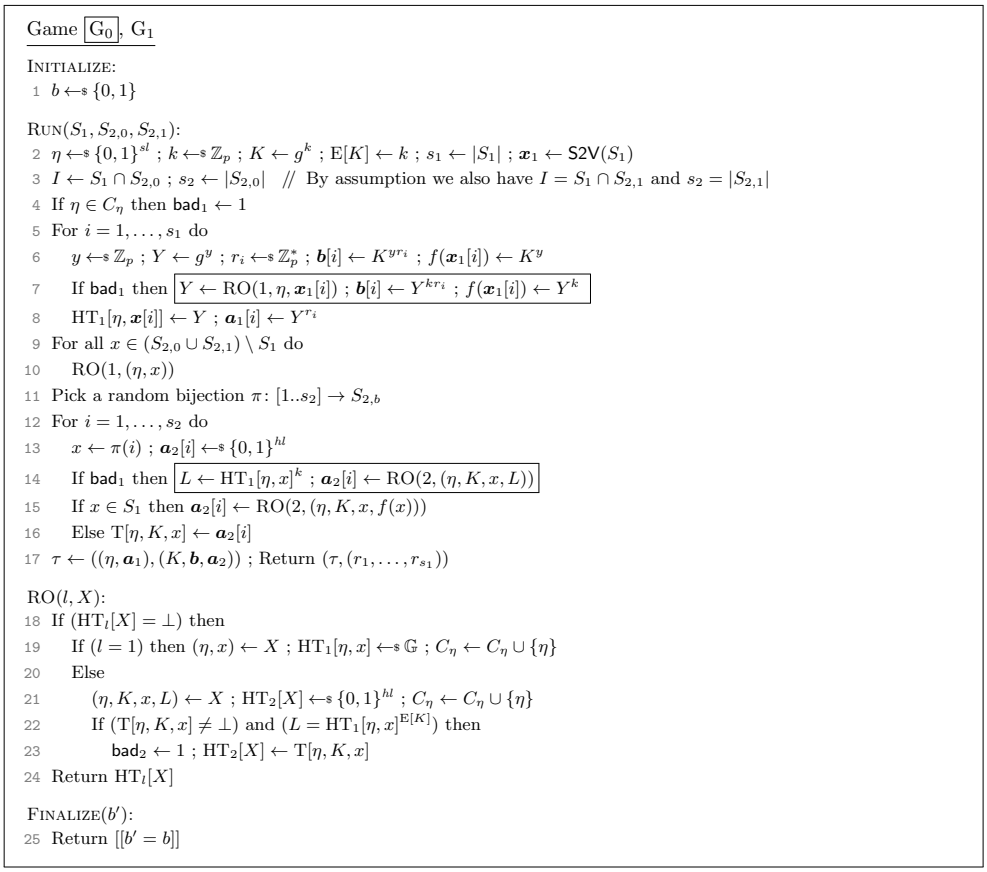


Figure 21: Games $\mathrm{G}_0, \mathrm{G}_1$ for the proof of Theorem 7.1. Game G_0 includes the boxed code and G_1 does not.

the client sends values of the form $X = \mathsf{H}_1(\eta, x)^r$. Let's think of η, x as fixed and known to the adversary. Using the same argument as in the proof of Theorem 6.1 given in Appendix J, let BD be the event that there is an x in the union of all client sets used in RUN queries such that $\mathrm{RO}(1, \eta, x)$ is the identity element of the group $\mathbb{G}$. If this bad event does not happen, then $w = \mathrm{RO}(1, \eta, x) \in \mathbb{G}^*$ is a generator for all $x \in S$, and thus w^r is uniformly distributed over $\mathbb{G}^*$ when $r \leftarrow\!\!\$ \mathbb{Z}_p^*$. Eq. (18) follows by observing that the probability of BD is at most $\mathrm{Q}^{\mathrm{RO}}(A_{\mathrm{ini}})$, where $\mathrm{Q}^{\mathrm{RO}}(A_{\mathrm{ini}})$ counts both the queries made directly by A_{ini} and the ones made by the protocol in the execution of the game with A_{ini}.

We now turn to the main claim, namely server security. As the theorem statement says, we will show different bounds on server security depending on the problem that is considered for the underlying group $\mathbb{G}$.

PROOF FOR xx $\in \{$v-cdh, v-cdh-muc$\}$. To make it simpler we show the reduction by creating an adversary $A_{\text{v-cdh-mu}}$ playing the V-CDH-MU game using A_{ini} and then use V-CDH $\rightarrow$ V-CDH-MU and V-CDH-MUC $\rightarrow$ V-CDH-MU to get the bounds claimed in Eq. (20). The idea is to start with a game that is the same as $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F},\Pi,2}$, but already incorporates conceptual changes to prepare for the reduction, and then move to a game where the adversary's view is independent of the challenge bit.

Let $q = \mathrm{Q}^{\mathrm{RUN}}(A_{\mathrm{ini}})$. Consider the games $\mathrm{G}_0, \mathrm{G}_1$ of Figure 21, where G_0 contains the boxed code and G_1 does not. Our first claim is that G_0 captures the InI game, meaning

$$\Pr[\mathbf{G}^{\mathrm{ini}}_{\mathsf{F},\Pi^{\mathrm{psi}},2}(A_{\mathrm{ini}})] = \Pr[\mathrm{G}_0(A_{\mathrm{ini}})] \; . \tag{30}$$

Let us explain and justify this. Throughout, for $l \in \{1,2\}$, table $\mathrm{HT}_l[\cdot]$ holds the value of $\mathsf{H}_l[\cdot]$. We create the games to allow NewKey outputs to be embedded as the keys used in Run oracle responses. Similarly, NewBase outputs will be embedded as the responses of $\mathsf{H}_1[\eta, x]$, but only for those pairs (η, x), where x is not in the client set for any Run oracle query. The set C_η stores all the salt values that have been queried to the random oracle. We will use this set to identify when programming the random oracle will fail.

Responses to the different Run queries use independent keys. This is done by picking a key at random at line 2. A random salt η is also chosen at line 2. However, if the salt has already appeared in any random oracle query, then a flag bad_1 is set in line 4. Now the game creates $\boldsymbol{a}_1, \boldsymbol{b}, \boldsymbol{a}_2$ for the transcript. For the latter two, the protocol would use the server key, here k. The intent of the game is to not use k so that later $K = g^k$ can play the role of the NewKey output. The vectors $\boldsymbol{a}_1$ and $\boldsymbol{b}$ need to be created using the elements in the client set. Thus, for these the game simply picks an exponent y and sets $\mathrm{HT}_1[\eta, \boldsymbol{x}_1[i]] = g^y$. This allows it to compute $\mathsf{H}_1(\eta, \boldsymbol{x}_1[i])^k$ as K^y which it needs to create $\boldsymbol{b}$. However, this programming is not possible if a query $\mathrm{RO}(1, (\eta, \boldsymbol{x}_1[i]))$ was made prior to this Run oracle query. This is where the salt will come in, ensuring (as we will show later) that this bad event has low probability. For now, at line 6, the game optimistically picks y and creates $Y = g^y$ as the intended value of $\mathrm{HT}_1[\eta, \boldsymbol{x}_1[i]]$. If the bad_1 flag has been previously set to true, then the boxed code reverts the optimistic choices to the correct ones, for this purpose using k directly. The result is that, whether or not bad_1 was set, $\boldsymbol{a}_1$ and $\boldsymbol{b}$ are correctly created in G_0.

The next step is to form $\boldsymbol{a}_2$ correctly while facilitating the embedding of NewBase oracle outputs. This is done via a combination of programming in Run and RO oracles. The loop at line 9 ensures that all elements in either of the server sets but not in the client set have the table entry $\mathrm{HT}_1[\eta, x]$ filled with a random group element (later used for NewBase challenges). To properly create $\boldsymbol{a}_2$, one must also randomly permute the elements of the server set as in the protocol. The game does this by explicitly picking π at line 11 based on which it creates $\boldsymbol{a}_2$ entries in the loop at line 12. If the bad_1 flag was previously set, then the boxed code of the game computes the entries of $\boldsymbol{a}_2$ by using k and querying the random oracle directly. It does the same for x that are in the client set since all the inputs for the random oracle query can be computed without k. Otherwise, a random value from appropriate range is picked at line 13 and stored in an additional table T. This is done so that later when a $\mathrm{RO}(2, (\eta, g^k, x, L))$ query is made such that $L = \mathsf{H}_1(\eta, x)^k$, the same response can be given. This is done at line 23, where we also set a flag bad_2. For now, this ensures that $\boldsymbol{a}_2$ is consistent with H_2 and completes our justification of Eq. (30).

Game G_1 simply drops the boxed code. The benefit from this is that it does not use k except for checking the condition at line 22. We will exploit this later. Note that we will first drop the code after bad_1 and in the next step we will look at bad_2. For now, we note that games $\mathrm{G}_0, \mathrm{G}_1$ are identical-until-bad_1, so by the Fundamental Lemma of Game Playing [15] we have

$$\Pr[\mathrm{G}_0(A_{\mathrm{ini}})] = \Pr[\mathrm{G}_1(A_{\mathrm{ini}})] + \Pr[\mathrm{G}_0(A_{\mathrm{ini}})] - \Pr[\mathrm{G}_1(A_{\mathrm{ini}})] \tag{31}$$

$$\leq \Pr[\mathrm{G}_1(A_{\mathrm{ini}})] + \Pr[\mathrm{G}_1(A_{\mathrm{ini}}) \text{ sets } \mathsf{bad}_1] \ . \tag{32}$$

The probability that game G_1 sets bad_1 is at most the probability that η, as chosen at line 2, arose in a prior RO query, which is at most $(\mathrm{Q}^{\mathrm{RO}}(A_{\mathrm{ini}}) + \mathrm{Q}^{\mathrm{Run}}(A_{\mathrm{ini}})) \cdot 2^{-sl}$. Taking the union bound over all Run queries, we get

$$\Pr[\mathrm{G}_1(A_{\mathrm{ini}}) \text{ sets } \mathsf{bad}_1] \leq \frac{\mathrm{Q}^{\mathrm{Run}}(A_{\mathrm{ini}}) \cdot \mathrm{Q}^{\mathrm{RO}}(A_{\mathrm{ini}}) + (\mathrm{Q}^{\mathrm{Run}}(A_{\mathrm{ini}}))^2}{2^{sl}} \ . \tag{33}$$

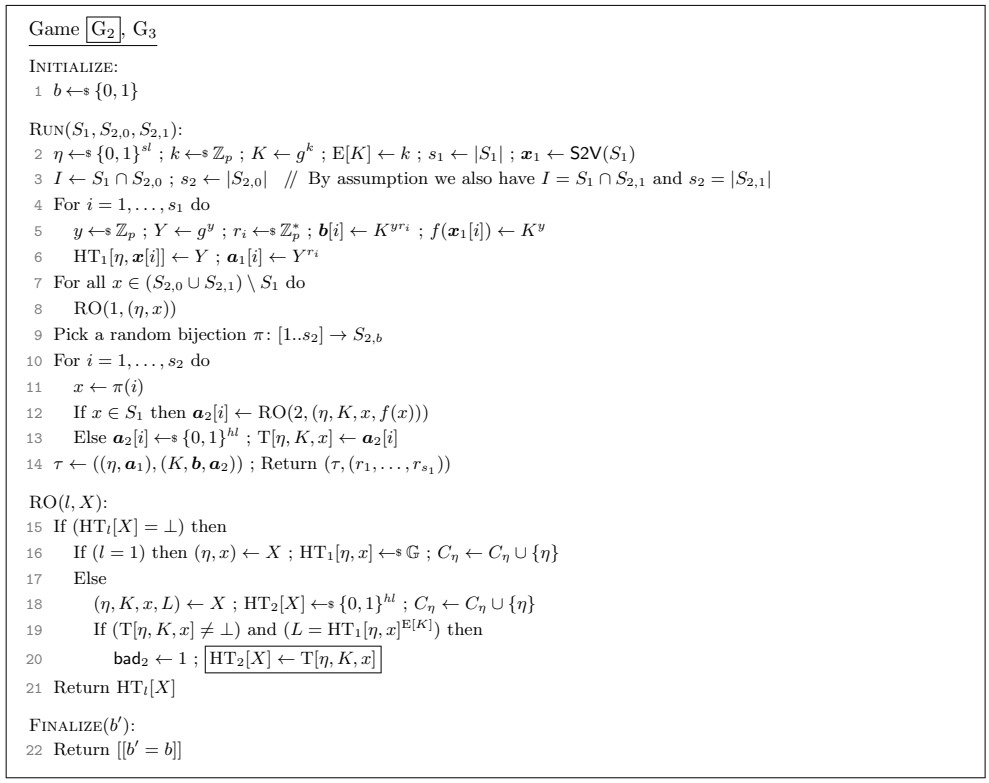


Figure 22: Games $\mathrm{G}_2, \mathrm{G}_3$ for the proof of Theorem 7.1. Game G_2 includes the boxed code and G_3 does not.

We now consider the games G_2 and G_3 in Figure 22, where the former includes the boxed code and the latter does not. We claim that

$$\Pr[\mathrm{G}_2(A_{\mathrm{ini}})] = \Pr[\mathrm{G}_1(A_{\mathrm{ini}})] \ , \tag{34}$$

which follows directly by observing that G_2 is obtained from G_1 by dropping unused code.

Finally, in G_3 we also drop the code after bad_2. The two games are identical-until-bad_2 and by the Fundamental Lemma of Game Playing [15] we have

$$\Pr[\mathrm{G}_2(A_{\mathrm{ini}})] = \Pr[\mathrm{G}_3(A_{\mathrm{ini}})] + \Pr[\mathrm{G}_2(A_{\mathrm{ini}})] - \Pr[\mathrm{G}_3(A_{\mathrm{ini}})] \tag{35}$$

$$\leq \Pr[\mathrm{G}_3(A_{\mathrm{ini}})] + \Pr[\mathrm{G}_3(A_{\mathrm{ini}}) \text{ sets } \mathsf{bad}_2] \ . \tag{36}$$

We bound $\Pr[\mathrm{G}_3(A_{\mathrm{ini}})$ sets $\mathsf{bad}_2]$ using the advantage of an adversary $A_{\text{v-cdh-mu}}$ playing the game $\mathbf{G}^{\text{v-cdh-mu}}_{\mathbb{G},g,p}$. To this end, we construct the $A_{\text{v-cdh-mu}}$ using A_{ini}. The adversary $A_{\text{v-cdh-mu}}$ picks a random bit b, initializes values i, i^*, j, j^* with 0 and Z with $\bot$ and runs A_{ini}. It answers oracle queries from A_{ini} using the same code as in the RUN and RO oracles of G_3 except some minor differences which we highlight. In line 2 instead of randomly picking k and then computing $K = g^k$, adversary $A_{\text{v-cdh-mu}}$ just makes a query as $K \leftarrow$ NEWKEY to get a random key. Note that since we are trying to simulate G_3, we don't need k. The value i is incremented every time a NEWKEY oracle call is made and $A_{\text{v-cdh-mu}}$ maintains a table T_1 to track the value i corresponding to a K.

The next difference comes at line 16 where instead of picking the group element for $\mathrm{HT}_1[\eta, x]$, the adversary $A_{\text{v-cdh-mu}}$ calls NEWBASE oracle and uses the output. At this point j is incremented and a table T_2 is used to track this j corresponding to (η, x).

The last difference is in line 19. Here the checks are performed differently, although with the same

goal. Using $(\eta, K, x, L) \leftarrow X$, adversary $A_{\text{v-cdh-mu}}$ gets $i' \leftarrow \mathrm{T}_1[K]$ and $j' \leftarrow \mathrm{T}_2[(\eta, x)]$. Then if i', j' are not $\perp$ it queries $\mathrm{DDHO}(i', j', L)$ to complete all the checks needed in line 19. If successful, it stores $i^* \leftarrow i'$, $j^* \leftarrow j'$ and $Z \leftarrow L$.

Finally, when A_{ini} completes, $A_{\text{v-cdh-mu}}$ returns (i^*, j^*, Z). It is easy to see that if $\mathrm{G}_3(A_{\text{ini}})$ sets bad_2 then A_{ini} wins its game. This gives us

$$\Pr[\mathrm{G}_3(A_{\text{ini}}) \text{ sets } \mathsf{bad}_2] \leq \mathbf{Adv}^{\text{v-cdh-mu}}_{\mathbb{G},g,p}(A_{\text{v-cdh-mu}}) \ .$$

Using V-CDH $\rightarrow$ V-CDH-MU and V-CDH-MUC $\rightarrow$ V-CDH-MU, we get

$$\Pr[\mathrm{G}_3(A_{\text{ini}}) \text{ sets } \mathsf{bad}_2] \leq \mathbf{Adv}^{\text{v-cdh}}_{\mathbb{G},g,p}(A_{\text{v-cdh}}) \ ,$$

$$\Pr[\mathrm{G}_3(A_{\text{ini}}) \text{ sets } \mathsf{bad}_2] \leq \mathbf{Adv}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}(A_{\text{v-cdh-muc}})$$

where $A_{\text{v-cdh}}$ and $A_{\text{v-cdh-muc}}$ are adversaries playing the games $\mathbf{G}^{\text{v-cdh}}_{\mathbb{G},g,p}$ and $\mathbf{G}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}$ respectively. The running time for $A_{\text{v-cdh}}$ increases by at most $(\mathrm{Q}^{\mathrm{RUN}}(A_{\text{ini}}) + M + \mathrm{Q}^{\mathrm{RO}}(A_{\text{ini}}))$ group exponentiations while it remains the same as $A_{\text{v-cdh-mu}}$ for $A_{\text{v-cdh-muc}}$.

Our final claim is that A_{ini} has no advantage in predicting the bit b in the game G_3, namely that

$$\Pr[\mathrm{G}_3(A_{\text{ini}})] = \frac{1}{2} \ . \tag{37}$$

We justify this claim using the following observation. The bit b is used to determine which server set ($S_{2,0}$ or $S_{2,1}$) to use to construct $\boldsymbol{a}_2$. This, in turn, means that any information about b that the adversary gets needs to come from $\boldsymbol{a}_2$ entries for elements $x \notin I$. For any such element, $\boldsymbol{a}_2$ entry is always a uniformly random value from $\{0,1\}^{hl}$. This means adversary gets no information about b, which gives us Eq. (37).

We now put the above equations together to conclude. For brevity, in the following, we let $g_i = \Pr[\mathrm{G}_i(A_{\text{ini}})]$; $\epsilon = \mathbf{Adv}^{\text{ini}}_{\mathsf{F},\Pi^{\text{psi}},2}(A_{\text{ini}})$; $\epsilon' \in \{\mathbf{Adv}^{\text{v-cdh}}_{\mathbb{G},g,p}(A_{\text{v-cdh}}), \mathbf{Adv}^{\text{v-cdh-muc}}_{\mathbb{G},g,p}(A_{\text{v-cdh-muc}})\}$; $\delta = \mathrm{Q}^{\mathrm{RUN}}(A_{\text{ini}}) \cdot (\mathrm{Q}^{\mathrm{RO}}(A_{\text{ini}}) + \mathrm{Q}^{\mathrm{RUN}}(A_{\text{ini}})) \cdot 2^{-sl}$. Then from the above we have

$$\epsilon = 2g_0 - 1 = 2(g_0 - g_1) + (2g_1 - 1) \leq 2\delta + (2g_2 - 1) \tag{38}$$

$$= 2\delta + 2(g_2 - g_3) + (2g_3 - 1) = 2\delta + 2\epsilon' + (2(1/2) - 1) = 2\delta + 2\epsilon' \tag{39}$$

which are the bounds for $\mathrm{xx} \in \{\text{v-cdh}, \text{v-cdh-muc}\}$.

Proof for xx = ddh. Using Theorem 5.2 and the above result for xx = v-cdh-muc, we get the claimed bound for xx = ddh.

Proof for $\mathrm{xx} \in \{\text{cdh}, \text{cdh-muc}\}$. Again, to make things easier, we construct an adversary $A_{\text{cdh-mu}}$ playing the CDH-MU game using A_{ini}. Then, using CDH $\rightarrow$ CDH-MU and CDH-MUC $\rightarrow$ CDH-MU we arrive at the claimed results. We will use the games $\mathrm{G}_0, \mathrm{G}_1, \mathrm{G}_2$ and G_3 from above for the analysis. The idea is similar as before. Using Eq. (30) through Eq. (37), we get

$$\Pr[\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi^{\text{psi}},2}(A_{\text{ini}})] \leq \frac{\mathrm{Q}^{\mathrm{RUN}}(A_{\text{ini}}) \cdot \mathrm{Q}^{\mathrm{RO}}(A_{\text{ini}}) + (\mathrm{Q}^{\mathrm{RUN}}(A_{\text{ini}}))^2}{2^{sl}} + \Pr[\mathrm{G}_3(A_{\text{ini}}) \text{ sets } \mathsf{bad}_2] + \frac{1}{2} \ . \tag{40}$$

Now we construct $A_{\text{cdh-mu}}$ using A_{ini}. This will allow us to bound the probability of A_{ini} setting bad_2 in G_3 using the advantage of $A_{\text{cdh-mu}}$ in winning $\mathbf{G}^{\text{cdh-mu}}_{\mathbb{G},g,p}$. Let $q = \mathrm{Q}^{\mathsf{H}}(A_{\text{ini}})$. Adversary $A_{\text{cdh-mu}}$ works similar to $A_{\text{v-cdh-mu}}$ from above with some differences arising due to the absence of DDHO oracle. We highlight the differences. Along with the values b, i, i^*, j, j^* and Z, $A_{\text{cdh-mu}}$ also picks a value $c^* \leftarrow\!\!\$\ \{1, \ldots, q\}$ and initializes another $c \leftarrow 0$. The value c is incremented in every

RO oracle query made by A_{ini}. When $\text{RO}(\ell, X)$ query turns ($c = c^*$) along with having ($\ell = 2$) and the input $(\eta, K, x, L) \leftarrow X$ such that $\text{T}_1[K] \neq \perp$ and $\text{T}_2[(\eta, x)] \neq \perp$, $A_{\text{cdh-mu}}$ makes $i^* \leftarrow \text{T}_1[K]$, $j^* \leftarrow \text{T}_2[(\eta, x)]$ and $Z \leftarrow L$. Finally, when A_{ini} completes $A_{\text{cdh-mu}}$ returns (i^*, j^*, Z).

Let E be the event that the c^*-th RO oracle query made by A_{ini} passes the conditions in line 19 (which would ensure $A_{\text{cdh-mu}}$ wins) of G_3. Then we have

$$\Pr[\text{G}_3(A_{\text{ini}}) \text{ sets } \mathsf{bad}_2 \cap E] \leq \mathbf{Adv}^{\text{cdh-mu}}_{\mathbb{G},g,p}(A_{\text{cdh-mu}})$$

$$\Pr[\text{G}_3(A_{\text{ini}}) \text{ sets } \mathsf{bad}_2] \cdot \Pr[\, E \mid \text{G}_3(A_{\text{ini}}) \text{ sets } \mathsf{bad}_2 \,] \leq \mathbf{Adv}^{\text{cdh-mu}}_{\mathbb{G},g,p}(A_{\text{cdh-mu}}) \,.$$

If $\text{G}_3(A_{\text{ini}})$ sets bad_2, A_{ini} makes at least one RO oracle query which passes the conditions in line 19. This means $\Pr[\, E \mid \text{G}_3(A_{\text{ini}}) \text{ sets } \mathsf{bad}_2 \,] \geq 1/q$. And so we get

$$\Pr[\text{G}_3(A_{\text{ini}}) \text{ sets } \mathsf{bad}_2] \leq q \cdot \mathbf{Adv}^{\text{cdh-mu}}_{\mathbb{G},g,p}(A_{\text{cdh-mu}}) \,.$$

Using CDH $\rightarrow$ CDH-MU, this gives us

$$\Pr[\text{G}_3(A_{\text{ini}}) \text{ sets } \mathsf{bad}_2] \leq q \cdot \mathbf{Adv}^{\text{cdh}}_{\mathbb{G},g,p}(A_{\text{cdh}}) \,, \tag{41}$$

for an adversary A_{cdh} playing the game $\mathbf{G}^{\text{cdh}}_{\mathbb{G},g,p}$. The running time of A_{cdh} increases by at most $(\text{Q}^{\text{RUN}}(A_{\text{ini}}) + M + \text{Q}^{\text{RO}}(A_{\text{ini}}))$ group exponentiations. And using CDH-MUC $\rightarrow$ CDH-MU, the above gives us

$$\Pr[\text{G}_3(A_{\text{ini}}) \text{ sets } \mathsf{bad}_2] \leq q \cdot \mathbf{Adv}^{\text{cdh-muc}}_{\mathbb{G},g,p}(A_{\text{cdh-muc}}) \,, \tag{42}$$

for an adversary A_{cdh} playing the game $\mathbf{G}^{\text{cdh}}_{\mathbb{G},g,p}$ with the same running time as $A_{\text{cdh-mu}}$.

Using Eq. (40) and Eq. (41), we get

$$\Pr[\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi^{\text{psi}},2}(A_{\text{ini}})] \leq q \cdot \mathbf{Adv}^{\text{cdh}}_{\mathbb{G},g,p}(A_{\text{cdh}}) + \frac{\text{Q}^{\text{RUN}}(A_{\text{ini}}) \cdot \text{Q}^{\text{RO}}(A_{\text{ini}}) + (\text{Q}^{\text{RUN}}(A_{\text{ini}}))^2}{2^{sl}} + \frac{1}{2}$$

$$\mathbf{Adv}^{\text{ini}}_{\mathsf{F},\Pi^{\text{psi}},2}(A_{\text{ini}}) \leq 2q \cdot \mathbf{Adv}^{\text{cdh}}_{\mathbb{G},g,p}(A_{\text{cdh}}) + 2 \cdot \frac{\text{Q}^{\text{RUN}}(A_{\text{ini}}) \cdot \text{Q}^{\text{RO}}(A_{\text{ini}}) + (\text{Q}^{\text{RUN}}(A_{\text{ini}}))^2}{2^{sl}} \,,$$

which is the bound for xx = cdh. And similarly using Eq. (40) and Eq. (42) we get,

$$\mathbf{Adv}^{\text{ini}}_{\mathsf{F},\Pi^{\text{psi}},2}(A_{\text{ini}}) \leq 2q \cdot \mathbf{Adv}^{\text{cdh-muc}}_{\mathbb{G},g,p}(A_{\text{cdh-muc}}) + 2 \cdot \frac{\text{Q}^{\text{RUN}}(A_{\text{ini}}) \cdot \text{Q}^{\text{RO}}(A_{\text{ini}}) + (\text{Q}^{\text{RUN}}(A_{\text{ini}}))^2}{2^{sl}} \,,$$

which is the bound for xx = cdh-muc. ∎