A preliminary version of this paper appears in the proceedings of Crypto 2026. This is the full version.

Just-in-Time-OPRFs and a Modular Framework for Fast Private Set Intersection

Mihir Bellare[1] Rishabh Ranjan[2] Doreen Riepel[3]

June 2026

Abstract

This paper gives a modular and unified framework within which to derive fast protocols for Private Set Intersection (PSI). At the core of this is a new primitive, that we define, and that we call a Just-In-Time OPRF (JIT-OPRF). We show how to obtain PSI generically from any JIT-OPRF, and then how to obtain JIT-OPRFs from Oblivious Transfer (OT) and Vector Oblivious Linear Evaluation (VOLE). We recover as special cases PSI protocols in the literature based on these two assumptions. Our results and proofs throughout are concrete rather than asymptotic, with explicit bounds that allow one to determine security parameters to achieve a desired level (e.g. 128 bits) of proven security in practice. Our results show interesting differences in the concrete security of OT and VOLE based PSI. Beyond the practical contribution of concrete-security, our work adds conceptual simplicity to this area, and opens the door to new PSI protocols via the construction of new JIT-OPRFs.

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

[3] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany. Email: riepel@cispa.de. URL: https://cispa.de/de/people/c01dori. Work partially done while at UCSD (supported in part by KACST).

Contents

1 Introduction **3**
- 1.1 The PSI landscape . . . 3
- 1.2 Our questions and goals . . . 3
- 1.3 Our answers and contributions . . . 6

2 Preliminaries **9**

3 2PC Definitional Framework **11**

4 JIT PRFs and JIT OPRFs **13**

5 PSI from JIT-OPRFs **16**
- 5.1 PSI from any JIT-OPRF . . . 16
- 5.2 Bin-allocation schemes . . . 18
- 5.3 PSI from JIT-OPRFs and Bin Allocation . . . 19

6 JIT-OPRF from OT **21**
- 6.1 Construction . . . 21
- 6.2 Security result . . . 23
- 6.3 Proof of Theorem 6.1 Part (1) . . . 24
- 6.4 Proof of Theorem 6.1 Part (2) . . . 25

7 JIT-OPRF from VOLE **32**
- 7.1 Construction . . . 32
- 7.2 Security result . . . 33
- 7.3 Proof of Theorem 7.1 Part (1) . . . 34
- 7.4 Proof of Theorem 7.1 Part (2) . . . 36

8 JIT-OPRF from KVS and VOLE **41**
- 8.1 Construction . . . 42
- 8.2 Security Result . . . 46
- 8.3 Proof for Theorem 8.1 Part (1) . . . 47
- 8.4 Proof for Theorem 8.1 Part (2) . . . 48
- 8.5 Universally Achieving EnUF Security . . . 54

References **57**

A Proof of Proposition 4.1 **60**

B Proof of Theorem 5.1 **61**
- B.1 Proof for Theorem 5.1 Part (1) . . . 61
- B.2 Proof for Theorem 5.1 Part (2) . . . 62
- B.3 Proof for Theorem 5.1 Part (3) . . . 63

C Proof of Theorem 5.2 **64**
- C.1 Proof for Theorem 5.2 Part (1) . . . 64
- C.2 Proof for Theorem 5.2 Part (2) . . . 67
- C.3 Proof for Theorem 5.2 Part (3) . . . 69

1 Introduction

1.1 The PSI landscape

What is PSI? Private set intersection (PSI) [38, 26, 18] is a popular two-party secure computation goal. Parties 1,2 (client and server, respectively) have, as their respective inputs, sets S_1, S_2. The functionality they want to securely compute is $\mathsf{F}^{\mathrm{psi}}(S_1, S_2) = (S_1 \cap S_2, \varepsilon)$. That is, at the end of the protocol, the client obtains $S_1 \cap S_2$ while the server obtains nothing. Security asks that the client not learn anything about $S_2 \setminus S_1$ and the server not learn anything about S_1.

Why is PSI important? The reason is its wide array of applications. PSI has been used in private contact discovery [37, 50, 33, 24], monitoring ad conversion rates [50, 28, 29], genome testing [3], botnet detection [41], proximity testing [44] and a lot more. PSI is currently deployed in browsers, by Google, Apple, Microsoft and others, to allow clients to check whether their credentials belong to a server-held set of breached ones [36, 1, 27, 40, 21].

How is security defined and measured? In the semi-honest (also called honest but curious) setting, parties are assumed to follow the protocol but then try to glean information about the input of the other party from their own view, meaning their coins, input and the interaction transcript. In the malicious setting, parties may deviate from the protocol. Many of the most practical protocols target the semi-honest setting. We, accordingly, also focus on this setting.

Security proofs for PSI protocols have historically targeted a simulation-based definition in the usual style of secure computation. However, simpler, indistinguishability-based definitions have also been given and considered [39, 7]. Importantly, BRRA [7] proved that their Input-Indistinguishability (InI) definition *is equivalent to* a simulation-based one for any two-party functionality satisfying a certain condition that is met by PSI. So for PSI, we can (and will) use the simpler InI.

Results and analyses have historically been asymptotic. However, concrete-security analyses, which give explicit bounds, are important to ensure that proofs support parameter values used in practice. Work in this direction for PSI was initiated in [7], and we will use their frameworks.

Fast PSI protocols. There are many possible approaches to accomplish PSI. The most efficient protocols, however, follow the OPRF-based PSI paradigm of Hazay and Lindell (HL) [23]. These protocols will be our focus.

Recall that in an Oblivious Pseudo-Random Function (OPRF) [43, 17, 32] protocol, the client has an input x, the server has a (secret) key K for a (regular) PRF G, and the protocol ends with them holding $\mathsf{G}(K, x)$ and ε, respectively, which we write succinctly as $(\mathsf{G}(K, x), \varepsilon) \leftarrow\!\!\$\ \mathrm{OPRF}_{\mathsf{G}}(x, K)$. To perform PSI, the server picks a key K and the parties run $(y_x, \varepsilon) \leftarrow\!\!\$\ \mathrm{OPRF}_{\mathsf{G}}(x, K)$ for all x in the client set S_1. The server now sends $Z = \{\, \mathsf{G}(K, x) \,:\, x \in S_2 \,\}$ to the client, who outputs $\{\, x \in S_1 \,:\, y_x \in Z \,\}$ as the intersection. But this takes $|S_1|$ evaluations of the OPRF, which is slow.

Improvements have come from two main approaches. The first is to use Oblivious Transfer (OT) extension [30], and the corresponding PSI protocols include [51, 50, 35, 46, 54, 16, 48, 14, 49]. The second is to use Vector Oblivious Linear Evaluation (VOLE) [2], and here the PSI protocols include [55, 19, 53, 10]. These protocols (which are our focus) are summarized in Figure 1.

1.2 Our questions and goals

We identify, and will address, the following issues and questions.

Concrete-security and proof tightness. Security proofs for PSI protocols have historically

Paper	Primitives / Assumption			
	OT	**VOLE**	**CH**	**Other**
PSZ14 [51], PSSZ15 [50]	✓		✓	
KKRT16 [35]	✓		✓	PRC, CrR
PRTY19 [48]	✓		✓	PRF, CrR
CM20 [14]	✓			PRF, CR, CrR
PRTY20 [49]	✓			KVS, RO
GPRTY21 [19], RS21 [55] RR22 [53], BPSY23 [10]		✓		KVS, RO

Figure 1: Summary of fast (OPRF-based) PSI protocols in the literature, showing the primitives used or assumptions made. Abbreviations used: OT: Oblivious Transfer, VOLE: Vector Oblivious Linear Evaluation, CH: Cuckoo Hashing, PRC: Pseudorandom Code, CrR: Correlation Robust Hash, PRF: Pseudorandom Function, CR: Collision Resistant Hash Function, KVS: Key-Value Store, RO: Random Oracle.

---

been asymptotic, meaning show that polynomial-time adversaries have negligible advantage in violating security of the PSI protocol, under a similar asymptotic hardness assumption on any underlying cryptographic primitives, for example the discrete logarithm (DL) problem. Implementations that evaluate performance however, use concrete parameters, such as a 256-bit elliptic curve group, where the discrete logarithm problem is thought to offer 128-bits of security. But how many bits s of security does the PSI protocol offer? The asymptotic proof of security for the PSI protocol gives no direct information about this, and does not preclude s being much smaller than 128, which could mean that the implemented PSI protocol is actually insecure. A concrete-security theorem and proof fills this gap. Given any adversary A with advantage (success probability) ϵ in violating security of the PSI protocol, the proof builds an adversary A$'$ of about the same running time as A, with an advantage ϵ' against the DL problem, such that $\epsilon \leq \mathbf{B}(\epsilon', \mathbf{R})$, for an explicit function $\mathbf{B}$, called the *bound*, which may also depend on some resources $\mathbf{R}$ of A, such as the number of queries to various oracles in its game. Common examples of bounds are $\mathbf{B}(\epsilon', \mathbf{R}) = \epsilon'$ or $\mathbf{B}(\epsilon', \mathbf{R}) = 2\epsilon'$ or $\mathbf{B}(\epsilon', \mathbf{R}) = q \cdot \epsilon'$ where $q \in \mathbf{R}$ is the number of queries to some oracle. From the bound, one can now determine how many bits s of security the implemented PSI protocol provides over the 256-bit elliptic curve, namely $s = 128$ or $s = 127$ or $s = 128 - \lg(q)$, respectively, for our three examples. The proof is said to be *tight* if $\mathbf{B}(\epsilon', \mathbf{R}) = c \cdot \epsilon' + \delta$ for some small constant c and small δ, like in the first two examples, in which case $s \approx 128$. To support the (happening) practical deployments of PSI, concrete security with tight proofs is thus crucial.

The value of concrete security and tight proofs is well understood in some domains of cryptography, such as symmetric and public-key cryptography, where there is a long tradition of work on these themes, but for secure computation and PSI, it is relatively new, and begins with BRRA [7]. They consider the canonical, above-described OPRF-based PSI construction of HL [23] and give a concrete-security proof of the InI-security of the PSI protocol, based on the security of the OPRF, with a tight reduction. To complete the circle, they also give tight security proofs for the 2H-DH OPRF [31].

We thus have a satisfactory concrete-security treatment of the basic PSI-from-OPRFs paradigm of HL [23]. However, the Figure 1 state-of-the-art PSI protocols of the last decade, while informally seen and discussed as "OPRF-based," do not, formally and strictly speaking, fall within the HL paradigm [23]. This is well understood by the works giving these protocols, who give their own direct and ad hoc (but asymptotic!) proofs. So the results of BRRA [7] do not apply to this set of PSI protocols. The first goal of our work is to fill this gap by giving concrete-security results, for

both the OT-extension and VOLE-based PSI protocols, with proofs as tight as possible.

Modular proofs and unification. The set of PSI protocols to consider is large [51, 50, 35, 48, 14, 49, 55, 19, 53, 10]. Giving a separate concrete-security proof for each would be tedious and somewhat unenlightening. This leads naturally to a question, which is also actually of independent interest in itself, namely whether there is some unifying paradigm or design approach that covers these protocols (including both the OT-extension based ones and the VOLE-based ones) and can be the basis for a modular proof approach.

What form might this take? All these works see themselves, at an informal level, as building some kind of OPRF and then using this to get PSI. It would be natural to try to formalize this, imagining a paradigm of the following form:

> **The X-OPRF-based PSI paradigm.** (1) One gives, not informally but as a formally-defined cryptographic primitive with a syntax and formal definition of security, some "OPRF-like" primitive, call it an "X-OPRF." (2) One gives a construction of PSI from any X-OPRF, and shows, formally and with as tight as possible a proof, that the PSI protocol is secure if the X-OPRF is secure, and (3) One gives, and proves secure, with proofs as tight as possible, constructions of X-OPRFs from OT-extension and VOLE, such that the PSI protocols emanating from this include the above-mentioned state-of-the-art ones from the literature as special cases.

First steps towards this were taken by the seminal work of KKRT [35]. They define BaRK-OPRFs (Batch Related-Key OPRFs), which can be seen as step (1) above with X = BaRK, and give an OT-extension-based BaRK-OPRF. They then design PSI using the method of PSSZ [50], but they do not prove security of the PSI protocol based on the assumed security of the BaRK-OPRF, meaning are missing part of step (2). BaRK-OPRFs also assume a very specific type of relation between keys and do not seem to cover other OPRF-like objects implicitly built in later works, including from VOLE, meaning do not allow (3).

We clarify that what we mean by OPRFs, the basic kind, are single input. RS [55] show how to construct PSI from multi-input OPRFs, meaning accomplish step (2), but again not all of the implicit OPRF constructions in the literature seem to meet the multi-input OPRF definition.

In conclusion, an X-OPRF-based PSI paradigm as described above, able in particular to derive, explain and prove the full set of state-of-the-art OPRF-based protocols, including both those from OT-extension and those from VOLE, thus putting all these under a common umbrella, does not yet exist. The second goal of our work is to fill this gap by giving a fully functional X-OPRF-based PSI paradigm incorporating all three of the above steps.

Filling gaps. There are gaps in proofs in prior work that we would like to fill.

The constructions in the literature mostly build PSI protocols with quadratic server running time. Optimizations are then used to convert this core PSI into an optimized PSI with linear server running time. The technique, beginning with PSZ [51, 52], is Cuckoo Hashing with a stash [47, 34]. However, proofs are given only for the core PSI. We are not aware of any work that proves security of the optimized and more efficient PSI. This is a gap that needs to be filled. As an indication that there is something to consider, note that Cuckoo hashing has a failure probability. The understanding in the PSI literature seems to be that this impacts only correctness, not security, but Yeo [61] points out that the latter is not true, giving an attack on the optimized PSI protocol that succeeds with the hashing failure probability. This calls for proofs that in particular reflect this loss. Our goal is thus proofs of the optimized PSI protocols with bounds that account for the security losses from the optimizations.

Towards this, we face again the fact that there are many different optimization methods (for example, using either 2 hash functions or 3). To avoid repeating analyses, we ask if the optimiza-

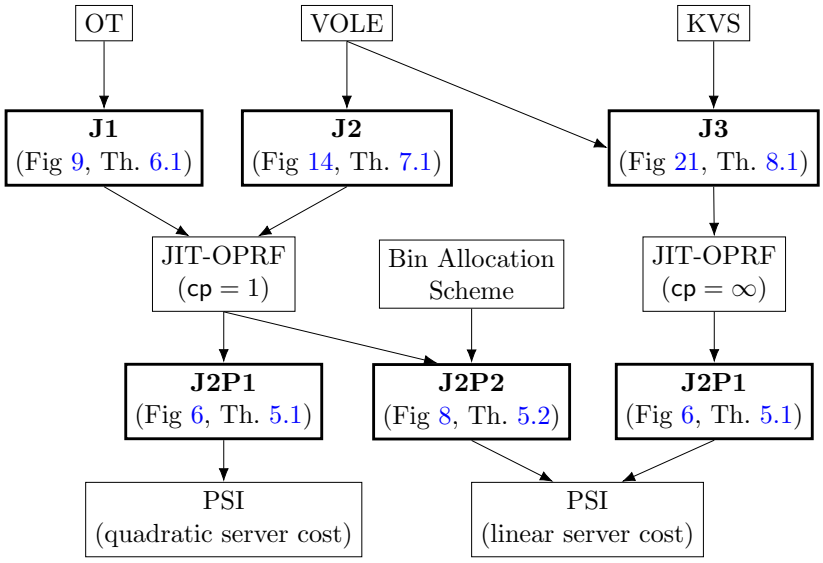


Figure 2: Our modular, unified framework for JIT-OPRF-based PSI. **J2P1**, **J2P2** are transforms that construct a PSI protocol from a JIT-OPRF, with server cost depending on the capacity $\mathsf{cp} \in \{1, \infty\}$ of the underlying JIT-PRF. **J1**, **J2**, **J3** are transforms that construct a JIT-OPRF from different ingredients, achieving capacities as shown.

---

tions too can be treated in a modular way, by first defining some abstract primitive, call it "Y", that captures them and provably yields the optimized PSI protocols, and then implementing Y in different ways. This would in particular open the door to new constructions of Y.

KKRT [35] use a PRC (Pseudorandom Code). They pick it at random and then fix it upfront as an input to the parties. But its definition gives a guarantee only if the code is chosen after, and independently of, the inputs to which it will be applied, and the latter are elements of the party sets. The result is that KKRT [35] is implicitly assuming party inputs do not depend on the PRC. Similarly, in using the Cuckoo-hashing optimization techniques discussed above, it is assumed that the inputs do not depend on the hash functions. We would like to lift these restrictions, giving protocols and security proofs for a setting where party inputs are dynamically and adaptively chosen by an adversary.

In summary, the third goal of our work is to fill the various gaps in prior PSI works discussed above.

1.3 Our answers and contributions

In this paper, we fully realize and achieve the X-OPRF paradigm. Through this, we arrive at concrete-security results, with proofs of varying tightness, for PSI based on OT-extension or on VOLE. Our framework covers not only the core, but also the optimized versions of the protocols.

The term X-OPRFs, above, is a variable, an object sought to make the paradigm work. A first and key step is to instantiate this variable, meaning give an actual primitive to play its role. We do this. We call our primitive a Just-In-Time OPRF (JIT-OPRF), and define it formally. This accomplishes step (1) of the paradigm. We then proceed to accomplish steps (2) and (3) with JIT-OPRFs in the role of X-OPRFs. The desired abstraction for the optimization methods, that we called Y, is another variable, which we instantiate with a primitive that we define and call a Bin Allocation Scheme. We now detail these components and steps. A summary diagram of our

framework is shown in Figure 2.

JIT-PRFs. We start by defining JIT PRFs, an extension of regular PRFs [20]. A JIT PRF f has a parameter $\mathsf{cp} \in \{1, \infty\}$ called the capacity that may be one of two things: either 1 (unit capacity) or ∞ (unbounded capacity). A JIT PRF, like a PRF, can be deterministically evaluated on a key $K \in \mathsf{Keys}$ and input $x \in \mathsf{Dom}$ via $y \leftarrow \mathsf{f.Ev}(K, x)$. The unusual aspect is key generation. The corresponding algorithm $\mathsf{f.Kg}$ takes any number $n > 0$ of inputs $x_1, \ldots, x_n \in \mathsf{Dom}$, and outputs a list $K_1, \ldots, K_m$ of keys. This allows the generation of multiple keys that can be arbitrarily related and dependent on some particular inputs. The value of m is n if $\mathsf{cp} = 1$ and 1 if $\mathsf{cp} = \infty$, meaning the capacity indicates how many inputs a single key can handle. Through a game shown in Figure 4, we define PRF security of JIT-PRFs.

JIT-PRFs extend BaRK PRFs [35], which can be captured as a special case of JIT PRFs in which the capacity is 1 and $K_i = (L, \ell_i)$ for $1 \le i \le n$ and random strings $L, \ell_1, \ldots, \ell_n$. JIT-PRFs however allow other and arbitrary types of relations between keys, and also allow keys to depend on inputs.

JIT-OPRFs. A JIT-OPRF is defined as a 2-party functionality $\mathsf{F}_{\mathsf{f}}^{\text{jit-oprf}}$ associated to a base JIT PRF f. As shown in Figure 4, it takes input $x_1, \ldots, x_n$ from party 1 (the client) and n from party 2 (the server). As output, the server gets $(K_1, \ldots, K_m) \leftarrow\!\!\$\ \mathsf{f.Kg}(x_1, \ldots, x_n)$ while the client gets the corresponding evaluations $y_1, \ldots, y_n$ where $y_i = \mathsf{f.Ev}(K_j, x_i)$ for $1 \le i \le n$ with $j = i$ if $\mathsf{cp} = 1$ and $j = m = 1$ if $\mathsf{cp} = \infty$. Note that unlike OPRFs, a JIT-OPRF is a randomized functionality due to key generation being randomized. JIT-OPRFs generalize, and capture as special cases, several prior OPRF-like primitives, including regular OPRFs, multi-input OPRFs, BaRK-OPRFs [35] and ci-rOPRFs (Client Independent Relaxed OPRFs) [58].

We must also of course define security of a protocol Π computing the JIT-OPRF functionality $\mathsf{F}_{\mathsf{f}}^{\text{jit-oprf}}$. As argued in [7], indistinguishability-style definitions, due to their single-quantifier nature, are more conducive to concrete security results than simulation-based definitions, which are double-quantifier. Accordingly, we give definitions in the former style. The one for client security, called JIT-OPRF1, requires that the keys $K_1, \ldots, K_m$ obtained by the server not reveal the client inputs $x_1, \ldots, x_n$. The one for server security, called JIT-OPRF2, extends the indistinguishability-based definition of OPRFs from [7], requiring PRF-like security, under $K_1, \ldots, K_m$, for client inputs different from $x_1, \ldots, x_n$. The games formalizing these are in Figure 5. Both definitions —as indeed with all our definitions— allow multiple executions of the protocol and are formulated with explicit adversary advantage functions to allow concrete-security results.

PSI from JIT-OPRFs: **J2P1.** We give a transform **J2P1** which constructs a PSI protocol from *any* JIT-OPRF protocol Π^{jit}. The construction follows the basic paradigm of prior work [23, 50, 35] with the extension that now the key(s) for the server can depend on the elements of the input set of the client. Theorem 5.1 proves client and server security of the PSI protocol $\mathbf{J2P1}[\Pi^{\text{jit}}]$ based, respectively, on the JIT-OPRF1 and JIT-OPRF2 security of the JIT-OPRF protocol Π^{jit}. The results are given with concrete-security bounds, and both proofs are tight. As above, to facilitate and simplify these analyses, the definition of security used for PSI is the Input-Indistinguishability (InI) one of [7], but we recall that the latter showed that InI is equivalent, for PSI, to a simulation-based definition, so nothing is lost.

The capacity cp of the JIT PRF f underlying Π^{jit} impacts the cost, specifically server running time and communication, of the PSI protocol **J2P1**. Namely, if f is unit capacity ($\mathsf{cp} = 1$) then the server cost is quadratic, like in the core PSI constructions in the literature, but if the capacity is unbounded ($\mathsf{cp} = \infty$) then the server cost is linear.

Bin allocation schemes. Our constructions of JIT-OPRFs from OT extension [30, 35] or VOLE,

discussed below, have unit capacity, and thus, under **J2P1**, yield PSI protocols with quadratic server running time. As in PSI work, we want to use techniques like Cuckoo hashing [47, 34] to reduce this cost. PSI works use these explicitly, but there are several variants. To abstract this process, we define in Section 5.2 a primitive we call a Bin Allocation Scheme. This instantiates the primitive we called Y above. As we illustrate, Cuckoo hashing is one way to build this, but there are other ways.

PSI FROM JIT-OPRFS: **J2P2**. We then give a second transform, **J2P2**, which constructs a PSI protocol from a unit capacity JIT-OPRF protocol Π^{jit} and a bin allocation scheme BS. The construction follows the basic paradigm of prior work [23, 50, 35]. The improvement over **J2P1** is that the server cost is linear (the constant depends on BS) even though the JIT-OPRF has unit capacity. Theorem 5.2 proves, as usual in a concrete-security setting, that this works, meaning the constructed PSI protocol is InI secure for both client and server. The proof is tight.

As discussed above, the literature is missing proofs for the optimized (Cuckoo hashing based) PSI protocols. Theorem 5.2 fills this gap with the first full proofs. Our result also explicitly shows the security loss from correctness error in the Bin Allocation scheme which, as noted above, was neglected in prior work.

In summary, we can obtain PSI with linear server running time in two ways: from a unit-capacity JIT-OPRF plus bin allocation, or directly from an unbounded capacity JIT-OPRF. We turn to building JIT-OPRFs that allow us to leverage these results.

A JIT-OPRF FROM OT: **J1**. Oblivious Transfer (OT) has been extensively used to build PSI [51, 50, 35, 16, 14, 49]. These constructions use the OT-extension technique of IKNP [30] for greater efficiency. We adapt these techniques to build a JIT-OPRF Π from any given OT protocol Π^{ot} (Figure 9), denoting by **J1** the transform that accomplishes this task. The capacity of the underlying JIT-PRF f is 1, so in obtaining PSI we would use **J2P2**.

As noted above, Pseudorandom Codes (PRCs) as used in [35] implicitly restrict party inputs to not depend on the code. Our JIT-OPRF1 and JIT-OPRF2 definitions however ask for a stronger form of security in which party inputs are chosen dynamically and adaptively by the adversary. To achieve this security, in place of a PRC, we use a random oracle [8]. Theorem 6.1 then shows both client (JIT-OPRF1) and server (JIT-OPRF2) security of Π, assuming the security of the underlying OT and using, for the latter, the InI definition of [7].

The proof for JIT-OPRF1-security is tight. For JIT-OPRF2-security, it is tight in the computational term, meaning the one for security of the OT protocol Π^{ot}. The statistical term (δ in Eq. (11)) depends on a statistical security parameter k which impacts the cost of the OT, so we want k to be as small as possible subject to δ being below a desired bound (say, 2^{-32}) on adversary advantage. Now δ in Eq. (11) has roughly the form $P \cdot 2^{-\gamma k}$ where P is a (quadratic) polynomial and γ is a constant. The good news is that δ degrades exponentially with k, and P is typical and should be viewed as small. However, we would ideally have $\gamma = 1$. Instead, it is smaller, which slows the rate of decrease. Section 6.2 gives estimates to see how this impacts the choice of k.

While more sophisticated analyses might improve this bound somewhat, our sense is that a non-ideal bound is essentially inherent in the construction techniques of prior work [30, 35] that we use. Very roughly, security here relies on the unpredictability, to the adversary, of a vector of the form $\mathbf{t} = \mathbf{a} \wedge \mathbf{s}$, where $\wedge$ is component-wise AND, $\mathbf{s} \in \{0,1\}^k$ is random and unknown to the adversary, and $\mathbf{a} \in \{0,1\}^k$ is known to the adversary and partially under its control. Now if a bit $\mathbf{a}[i] = 0$, then $\mathbf{t}[i] = 0$ is immediately predictable, so unpredictability of $\mathbf{t}$ requires that the Hamming weight of $\mathbf{a}$ not be too small. This explains the terms of the form $2^{-\gamma k}$ with $\gamma < 1$ in the bound. See Section 6.4. We note that we can show a better bound for our VOLE-based JIT-OPRF, discussed next.

Combining Theorems 5.1 and 6.1 yields a full proof of the security of the basic KKRT PSI protocol, with explicit bounds, and likewise combining Theorems 5.2 and 6.1 yields the same for the optimized version of the KKRT PSI protocol.

A JIT-OPRF FROM VOLE: **J2**. More recently, fast PSI is being built from Vector Oblivious Linear Evaluation (VOLE) [55, 19]. We adapt these techniques to build a JIT-OPRF Π from any given VOLE protocol Π^{v} (Figure 14), denoting by **J2** the transform that accomplishes this task. The capacity of the underlying JIT-PRF f is 1, so in obtaining PSI we would use **J2P2**. Theorem 7.1 shows both client (JIT-OPRF1) and server (JIT-OPRF2) security of Π, assuming the security of the underlying VOLE protocol and using, for the latter, the InI definition of [7].

The proof for JIT-OPRF1-security is tight, as with **J1**. Unlike for the latter, however, it is also tight for JIT-OPRF2-security. To elaborate, VOLE works over a field $\mathbb{F}$. To compare with **J1**, let k be the statistical security parameter in the latter, and select $\mathbb{F} = \mathrm{GF}(2^k)$ for **J2**. Then the statistical term (δ in Eq. (23)) in part 2 of Theorem 7.1 has the form $P \cdot 2^{-k}$ for a quadratic polynomial P. In particular there is no factor $\gamma < 1$ in the exponent, making this term ideal. Section 7.2 gives estimates that show that, for the same adversary resources and desired bound on advantage, k can be picked much smaller for **J2** than for **J1**. The result is that, for a common, desired level of security, the former is faster than the latter.

A JIT-OPRF FROM VOLE AND KVS: **J3**. Optimizing PSI protocols using a Linear Key Value Store (LKVS) was first introduced in [49], with [19] formally defining LKVSs and giving improvements. In Section 8 we adapt these techniques to build a JIT-OPRF Π from any given VOLE protocol Π^{v} and LKVS, denoting by **J3** the transform that accomplishes this task. The improvement here is that Π has unbounded capacity and thus the simpler **J2P1** transform can be used to obtain PSI. (In contrast, **J1** and **J2** construct unit-capacity JIT-OPRFs, and one has to use **J2P2** for PSI.) We show both client (JIT-OPRF1) and server (JIT-OPRF2) security of Π, assuming the security of the underlying VOLE protocol and using, for the latter, the InI definition of [7]. For the latter, we need a new property of LKVSs, that we define and call encoding unforgeability (EnUF). In Section 8.5, we give a cheap way to add EnUF security to any LKVS scheme. The transform is information-theoretic, meaning no computational assumptions are involved.

Similar to the security results of previous JIT-OPRF protocols, the JIT-OPRF1-security is tight. The bound for JIT-OPRF2-security again contains a computational and statistical term. Both of which are tight, as in **J2**.

OTHER PSI PROTOCOLS? Our main focus has been the fast, OPRF-based PSI protocols of Figure 1, and we have arrived at concrete-security proofs for them through our unified framework. It turns out one can also use JIT-OPRFs as an abstraction for other PSI paradigms including protocols based on polynomial evaluation [18] and key agreement [56], but we do not pursue this here. Our constructions do not cover the PSI protocols in [48, 14], and extending our work to do so (by giving corresponding JIT-OPRFs) would be interesting future work. More broadly, we clarify that our framework does not cover all, or all possible, PSI protocols; that is not the intent.

CORRECTION TO PROCEEDINGS VERSION. The bounds in Theorems 6.1 and 7.1 in this version of the paper correct and update the ones from the preliminary version [6].

2 Preliminaries

GENERAL NOTATION. 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 statement 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 $\{0,1\}^*$.

SETS, VECTORS AND MATRICES. If D is a finite set, then $|\mathrm{D}|$ denotes its size and $x \leftarrow\!\!\$\, \mathrm{D}$ denotes picking x uniformly at random from D. If $\mathbf{w}$ is a vector then $|\mathbf{w}|$ is its length (the number of coordinates) and $\mathbf{w}[i]$ is its i-th coordinate. We say that $\mathbf{w}$ is an n-vector if $|\mathbf{w}| = n$. We say $\mathbf{w}$ is a vector over a set S if $\mathbf{w}[i] \in S$ for all $1 \le i \le |\mathbf{w}|$. Strings are identified with vectors over $\{0,1\}$, so that $|Z|$ denotes the length of a string Z and $Z[i]$ denotes its i-th bit. By ε we denote the empty string or vector. By $x \| y$ we denote the concatenation of strings x, y. For equal-length vectors $\mathbf{x}_0, \mathbf{x}_1$ over a field, the inner product is denoted $\langle \mathbf{x}_0, \mathbf{x}_1 \rangle$.

We will employ several conversions between vectors and sets. We let $\mathsf{V2S}(\mathbf{w})$ (vector to set) be the set $\{\mathbf{w}[1], \ldots, \mathbf{w}[|\mathbf{w}|]\}$ of elements of vector $\mathbf{w}$. Likewise, if S is a set, then $\mathbf{w} \leftarrow \mathsf{S2V}(S)$ (set to vector) puts the elements of S into a vector in some canonical order, say lexicographic.

When multiplying matrices by vectors, it is important to be clear whether the vector is a row or column one. To this end, if $\mathbf{v} \in \mathrm{D}^\ell$ is a vector of length ℓ, we may write $\mathbf{v} \in \mathrm{D}^{1\times\ell}$ to emphasize that it is viewed as a row vector and $\mathbf{v} \in \mathrm{D}^{\ell\times 1}$ to emphasize that it is viewed as a column vector. By $\mathrm{D}^{m\times n}$ we denote the set of m by n matrices over a set D. If $\mathbf{A} \in \mathrm{D}^{m\times n}$ then the row i column j entry of $\mathbf{A}$ is denoted $\mathbf{A}[i][j]$, the i-th row of $\mathbf{A}$ is $\mathbf{A}[i][\cdot] \in \mathrm{D}^{1\times n}$ and the j-th column is $\mathbf{A}[\cdot][j] \in \mathrm{D}^{m\times 1}$. If $\mathbf{A} \in \mathrm{D}^{m\times n}$ and $\mathbf{B} \in \mathrm{D}^{n\times k}$ for a field D, then $\mathbf{A}\cdot\mathbf{B} \in \mathrm{D}^{m\times k}$ denotes their matrix product. If $\mathbf{s}$ is a k-vector over D then $\mathbf{diag}(\mathbf{s}) \in \mathrm{D}^{k\times k}$ denotes the matrix defined by $\mathbf{diag}(\mathbf{s})[i][i] = \mathbf{s}[i]$ and $\mathbf{diag}(\mathbf{s})[i][j] = 0$ if $i \ne j$.

ORACLE SPACES AND RANDOM ORACLES. In the random oracle model [8], the domain and range of the random oracle can depend on the scheme. (The latter term here includes protocols and functionalities.) Accordingly, following [7], 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 $\mathrm{RO}(X)$ that 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 [9]. A game G specifies an INITIALIZE procedure, further procedures (also called oracles) and a FINALIZE procedure. 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 $\Pr[\mathrm{G}(\mathrm{A}) \Rightarrow y]$ we denote the probability that the execution of game G with adversary A results in the game output being y, and write just $\Pr[\mathrm{G}(\mathrm{A})]$ for $\Pr[\mathrm{G}(\mathrm{A}) \Rightarrow 1]$. 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. We write $\mathrm{Q}^{\mathrm{O}}(\mathrm{A})$ to denote the number of queries A makes to the oracle O. The running time $\mathrm{T}(\mathrm{A})$ of adversary A is defined as the running time of the execution of the game with the adversary, so that it includes the time taken by game oracles to respond to adversary oracle queries. This convention means that reductions mostly preserve running time.

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 [9].

Lemma 2.1 [Fundamental Lemma of Game Playing [9]] *Let* G, H *be identical-until-*bad *games. Then for any adversary* A *we have*

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

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.

3 2PC Definitional Framework

We recall the BRRA definitions of syntax and security for two party computation in a concrete setting [7], in particular the indistinguishability-based InI definition of security. We see the party identities as $1, 2$ with 1 being the "client" and 2 being the "server."

Functionalities. A (two-party) functionality describes the function that the parties want to compute. Formally, it is an algorithm F that takes two inputs and provides two outputs corresponding to the two parties via $(y_1, y_2) \leftarrow\!\!\$\ \mathsf{F}[\mathsf{H}](x_1, x_2)$. Here, $\mathsf{H} \in \mathsf{OS}$ is given as oracle to F. A functionality can be deterministic or randomized and the above covers randomized functionalities. Note a deterministic functionality is a special case of a randomized one and so the same notation applies, but in this case we may, for emphasis, also write $(y_1, y_2) \leftarrow \mathsf{F}[\mathsf{H}](x_1, x_2)$.

PSI functionality. The PSI functionality $\mathsf{F}^{\mathrm{psi}}$ is defined as:

$\underline{\mathsf{F}^{\mathrm{psi}}((S_1, s_2), (S_2, s_1))}$ // $S_1, S_2 \subseteq \{0,1\}^*$ and $s_1 = |S_1|$ and $s_2 = |S_2|$
$I \leftarrow S_1 \cap S_2$; Return (I, ε)

This functionality is deterministic. Party $i \in \{1, 2\}$ has input a set $S_i \subseteq \{0,1\}^*$ and an integer s_{3-i}. It is required that $s_{3-i} = |S_{3-i}|$, meaning the second input of each party is the size of the set of the other party. PSI protocols, particularly the ones that we build, tend to leak set sizes anyway, so this requirement, while convenient for us, is not usually limiting. As output, party 1 gets the intersection $I = S_1 \cap S_2$ while party 2 gets nothing (formally the empty string ε).

Protocols. 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

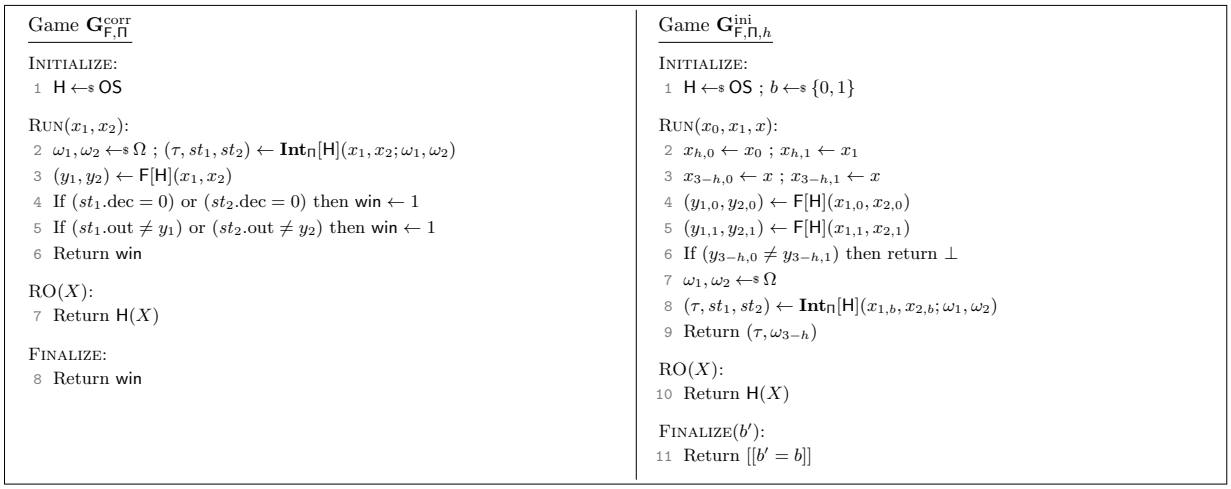


Figure 3: **Left:** Game defining correctness for protocol Π for deterministic functionality F. **Right:** Game defining InI security for protocol Π for functionality F, where $h \in \{1, 2\}$ is the honest party.

$st.\mathrm{dec} \in \{1, 0\}$. The interaction consists of $\mathsf{nr} \in \mathbb{N}$ moves (also called rounds). By convention, 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 the parties and access to an oracle $\mathsf{H} \in \mathsf{OS}$ to generate an execution trace $(\tau, st_1, st_2) \leftarrow \mathbf{Int}_{\Pi}[\mathsf{H}](x_1, x_2; \omega_1, \omega_2)$. 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:

$\underline{\mathbf{Int}_{\Pi}[\mathsf{H}](x_1, x_2; \omega_1, \omega_2)}$
$st_1.\mathrm{in} \leftarrow x_1$; $st_2.\mathrm{in} \leftarrow x_2$; $m_0 \leftarrow \varepsilon$; $i \leftarrow 1$
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. We use the game-based correctness definition of [7] shown in Figure 3. For a protocol Π for a functionality F, the correctness advantage of an adversary $\mathrm{A}_{\mathrm{corr}}$ is given as $\mathbf{Adv}^{\mathrm{corr}}_{\mathsf{F},\Pi}(\mathrm{A}_{\mathrm{corr}}) = \Pr[\mathbf{G}^{\mathrm{corr}}_{\mathsf{F},\Pi}](\mathrm{A}_{\mathrm{corr}})$. The game provides a Run oracle which $\mathrm{A}_{\mathrm{corr}}$ can use to execute Π on inputs (x_1, x_2). It wins if either the parties reject or their outputs do not match those of the functionality. Multiple calls to Run are allowed. The inputs in these calls are required to be valid for the functionality. (For example, for PSI, $x_1 = (S_1, s_2)$ and $x_2 = (S_2, s_1)$ where $s_j = |S_j|$ and $S_j \subseteq \{0,1\}^*$ for $j \in \{1, 2\}$.)

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 recall the single-quantifier indistinguishability-style definition called *input indistinguishability* (InI) of [7] below. Recall that for some functionalities, including PSI, BRRA [7] showed that InI is equivalent to a simulation-based definition, so there is no loss of security in working with InI.

The following assumes wlog that the RO spaces of functionality F and a protocol Π for it are the same. (One can always work with an appropriate union of the two spaces if not.)

INI DEFINITION. The definition assumes the functionality F is deterministic. (InI has so far not been defined for randomized functionalities.) The game, denoted $\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,h}$, is in Figure 3. The ini-advantage of an adversary A_{ini} is then defined by $\mathbf{Adv}^{\text{ini}}_{\mathsf{F},\Pi,h}(\mathrm{A}_{\text{ini}}) = 2\Pr[\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,h}(\mathrm{A}_{\text{ini}})] - 1$. 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 of inputs for the honest party and a single choice x for the corrupt party. It is required that these inputs are valid for the functionality. (For example, for PSI, if $h = 1$, then $x_0 = (S_{1,0}, s_2)$ and $x_1 = (S_{1,1}, s_2)$ and $x = (S_2, s_1)$ where $s_2 = |S_2|$ and $s_1 = |S_{1,0}| = |S_{1,1}|$ and $S_{1,0}, S_{1,1}, S_2 \subseteq \{0,1\}^*$.) This results in two pairs of inputs for the functionality. At lines 4-5 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.

4 JIT PRFs and JIT OPRFs

We define an extension of OPRFs [43, 17] and BaRK-OPRFs [35] that we call *Just in Time* (JIT) OPRFs. We then show how to build PSI generically from any JIT-OPRF. We give many constructions of JIT-OPRFs, including from OT-extension and VOLE. Our results and proofs, throughout, are in a concrete-security setting. As a result of this, we capture and unify many efficient PSI protocols [35, 48, 49, 14, 19, 55, 53, 10], which emerge as JIT-OPRF-based PSI instantiated with a particular JIT-OPRF, and obtain concrete-security analyses of them.

JIT-PRFs. We begin by defining an extension of PRFs [20] and RK-PRFs [35] that we call *Just in Time* (JIT) PRFs to reflect that keys are input-dependent, and generated on the fly at the time they are needed. Starting with syntax, a *JIT family of functions* f defines parameters and algorithms as follows:

- $\mathsf{cp} \in \{1, \infty\}$— This parameter is called the *capacity* of the JIT family of functions, and takes value either 1 (unit capacity) or ∞ (unbounded capacity). A generalization to allow the capacity to be any positive integer is possible but not necessary here.
- $y \leftarrow \mathsf{Ev}[\mathsf{H}](K, x)$— Deterministic evaluation algorithm $\mathsf{Ev}\colon [\mathsf{OS}] \times \mathsf{Keys} \times \mathsf{Dom} \to \mathsf{Rng}$ takes a key $K \in \mathsf{Keys}$ and input $x \in \mathsf{Dom}$ and, with oracle access to $\mathsf{H} \in \mathsf{OS}$, returns an output y.
- $(K_1, \ldots, K_{\mathsf{nk}(\mathsf{cp},n)}) \leftarrow\!\!\$\ \mathsf{Kg}[\mathsf{H}](x_1, \ldots, x_n)$— Key generation algorithm Kg takes any number $n \geq 1$ of points $x_1, \ldots, x_n \in \mathsf{Dom}$ to return a list of $\mathsf{nk}(\mathsf{cp}, n)$ keys $K_1, \ldots, K_{\mathsf{nk}(\mathsf{cp},n)} \in \mathsf{Keys}$, where the function $\mathsf{nk}(\mathsf{cp}, \cdot)\colon \mathbb{N} \to \mathbb{N}$ is defined simply by $\mathsf{nk}(1, n) = n$ and $\mathsf{nk}(\infty, n) = 1$. That is, when the capacity is 1, there is one key per input, and when the capacity is ∞, there is a single key across all inputs. Note that the key generation algorithm can take any number $n \geq 1$ of inputs, and this number can vary across calls if desired.

As usual, the random oracle H is drawn from some oracle space OS associated to f. For emphasis or disambiguation, we may write $\mathsf{f.cp}, \mathsf{f.Ev}, \mathsf{f.Kg}, \mathsf{f.nk}, \mathsf{f.Dom}, \mathsf{f.Rng}, \mathsf{f.OS}$ for the different subcomponents of f. The capacity, intuitively, is the number of inputs that can be handled by a single key.

Figure 4 shows the game defining JIT-PRF security of f. The adversary A can create a vector $(K_1, \ldots, K_{\mathsf{f.nk}(\mathsf{cp},n)})$ of keys via a call to oracle GENKEY. The oracle returns nothing. The adversary can also call $\mathrm{FN}(i', j, x)$ for any $x \in \mathsf{Dom}$. Depending on the challenge bit c, the reply is either determined via Ev or chosen at random. The advantage of A is $\mathbf{Adv}^{\text{jit-prf}}_{\mathsf{f}}(\mathrm{A}) = 2\Pr[\mathbf{G}^{\text{jit-prf}}_{\mathsf{f}}(\mathrm{A})] - 1$.

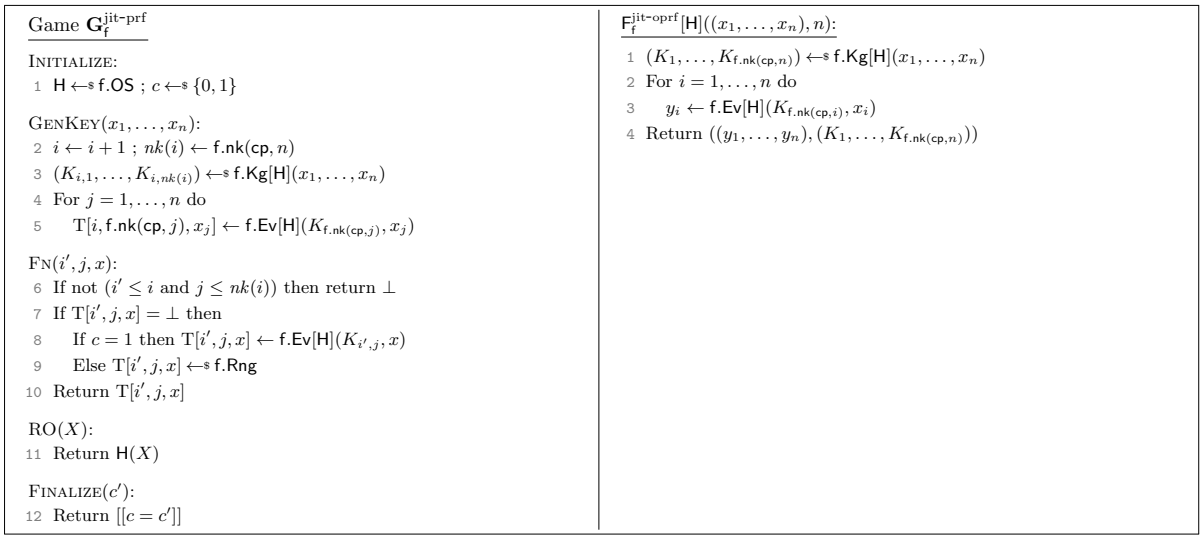


Figure 4: **Left:** Game defining JIT-PRF security for JIT function family f. **Right:** Oblivious JIT-PRF functionality $\mathsf{F}_{\mathsf{f}}^{\text{jit-oprf}}$ associated to JIT function family f.

We recover standard PRFs as the special case of JIT PRFs in which the capacity is unbounded and Kg ignores its inputs. The JIT-PRF definition then captures multi-user PRF security, seeing each i as a user with key $K_{i,1}$ that is initialized via the i-th call to GenKey.

JIT-OPRF functionality. Let f be a JIT family of functions with $\mathsf{f.Ev}\colon [\mathsf{OS}] \times \mathsf{Keys} \times \mathsf{Dom} \to \mathsf{Rng}$. We associate to it a functionality $\mathsf{F}_{\mathsf{f}}^{\text{jit-oprf}}$ that we call the JIT Oblivious PRF (JIT-OPRF) functionality for f. It has oracle space OS. The client's input is a list $x_1, \ldots, x_n$ of vectors over Dom. The server's input is n. The functionality is randomized. As shown in Figure 4, it generates keys via $K_1, \ldots, K_{\mathsf{f.nk}(\mathsf{cp},n)} \leftarrow\!\!\$\ \mathsf{Kg}[\mathsf{H}](x_1, \ldots, x_n)$. It then evaluates the base JIT-PRF f on the elements of the vectors to obtain vectors $y_1, \ldots, y_n$ over Rng. This list is returned to Party 1. Party 2 gets the keys $K_1, \ldots, K_{\mathsf{f.nk}(\mathsf{cp},n)}$, allowing it, for any $K \in \mathsf{V2S}(K_1, \ldots, K_{\mathsf{f.nk}(\mathsf{cp},n)})$, to evaluate $\mathsf{Ev}[\mathsf{H}](K, x)$ on any x of its choice.

The above formulation can be seen as a generalization of the OPRF functionality as one can use an OPRF protocol for an underlying PRF f to construct a JIT-OPRF protocol for f. Note that as mentioned above any PRF is also a JIT-PRF.

JIT-OPRF correctness. We now define correctness for a protocol Π for a JIT-OPRF based on a JIT family of functions f with capacity cp. We say Π correctly realizes $\mathsf{F}_{\mathsf{f}}^{\text{jit-oprf}}$ if, for all possible inputs $x_1, \ldots, x_n \in \mathsf{f.Dom}$, where $n \ge 1$, and for any choice of $\mathsf{H} \in \mathsf{OS}$, the following is true. Suppose $y_1, \ldots, y_n$ and $(K_1, \ldots, K_{\mathsf{nk}(\mathsf{cp},n)})$ are outputs for party 1 and party 2, respectively, from executing Π on input $x_1, \ldots, x_n$ for party 1 and n for party 2. Then, for all $i \in [1..n]$ we have $y_i = \mathsf{f.Ev}[\mathsf{H}](K_{\mathsf{f.nk}(\mathsf{cp},i)}, x_i)$ and $K_i \in \mathsf{f.Keys}$.

We could also ask that $(K_1, \ldots, K_{\mathsf{nk}(\mathsf{cp},n)})$ is distributed identically to the output of $\mathsf{f.Kg}[\mathsf{H}](x_1, \ldots, x_n)$, but this, although true for our constructions, tends to be unnecessary to require explicitly. The definitions of security provide whatever is needed instead.

JIT-OPRF security. Let f be a JIT family of functions and $\mathsf{F}_{\mathsf{f}}^{\text{jit-oprf}}$ the associated JIT-OPRF functionality. Let Π be a protocol for $\mathsf{F}_{\mathsf{f}}^{\text{jit-oprf}}$. We will give separate definitions for JIT-OPRF security for client and server, similar to the OPRF security definitions in [7].

In the left panel of Figure 5, we give our JIT-OPRF1 definition for client-side security of a JIT-OPRF protocol Π. This indistinguishability game provides a CH oracle which takes a pair

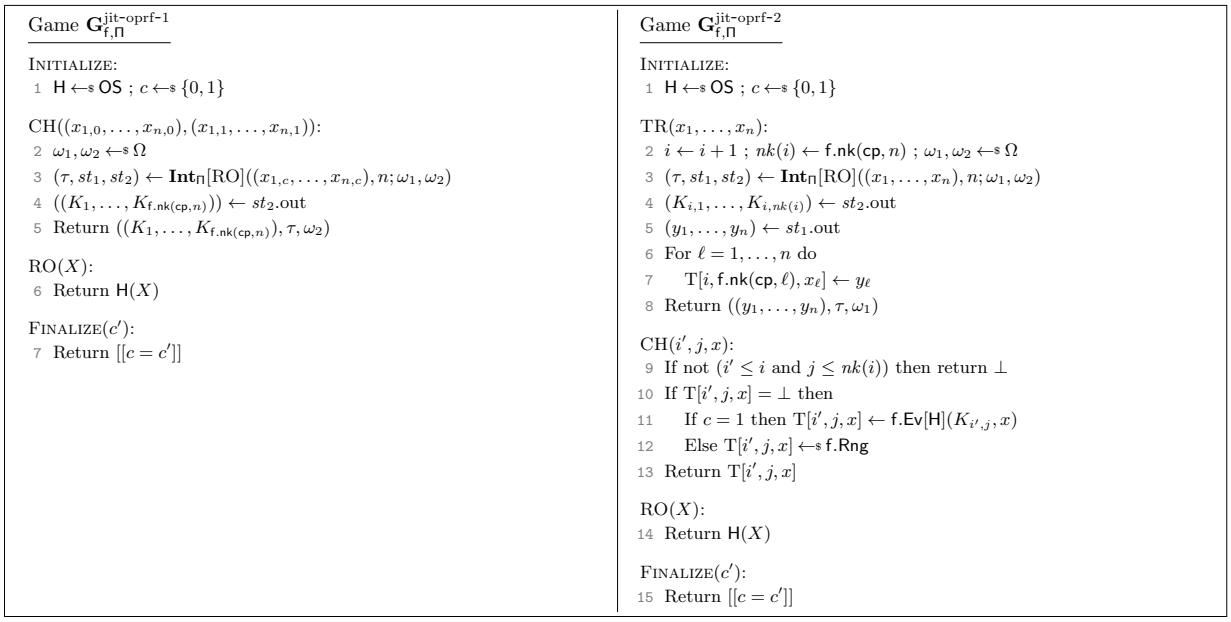


Figure 5: JIT-OPRF1 game for client side security and JIT-OPRF2 game for server side security of a JIT OPRF protocol Π.

of inputs $(x_{1,0},\ldots,x_{n,0}),(x_{1,1},\ldots,x_{n,1})$ for the client. Note that the number of values, n, is same for both inputs since it is the input to the functionality for the dishonest party (server). The CH oracle picks coins for both parties and then executes Π on one of the inputs determined by the challenge bit c and returns the transcript τ along with the output and random coins of the server $(K_1,\ldots,K_{\mathsf{f.nk}(\mathsf{cp},n)},\omega_2)$. An adversary A playing the game has access to the CH and RO oracles and wins if it correctly guesses the challenge bit c. The advantage for A is given as $\mathbf{Adv}^{\text{jit-oprf-1}}_{\mathsf{f},\Pi}(\text{A}) = 2 \cdot \Pr[\mathbf{G}^{\text{jit-oprf-1}}_{\mathsf{f},\Pi}(\text{A})] - 1$.

For server-side security of JIT-OPRF protocols we define JIT-OPRF2 as an indistinguishability from random game in Figure 5. The game first initializes a random bit c and a random oracle H. It provides a TR oracle which on input $(x_1,\ldots,x_n)$ executes Π to return the output $y_1,\ldots,y_n$ and coins ω_1 for party 1 along with the transcript τ. The i-th TR oracle query generates a vector K_i of keys and the adversary can then use the CH oracle with inputs (i', j, x) to either get $\mathsf{f.Ev}(K_{i',j}, x)$ (when $c = 1$) or a uniformly sampled value from $\mathsf{f.Rng}$ (when $c = 0$). The adversary wins if it can correctly guess the challenge bit c. We let $\mathbf{Adv}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi}(\text{A}) = 2\Pr[\mathbf{G}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi}(\text{A})] - 1$ be the advantage of an adversary A.

It is easy to see that JIT-OPRF2 security of a JIT-OPRF protocol Π^{jit} for a JIT function family f implies JIT-PRF security of f. This is formalized in Proposition 4.1 with proof given in Appendix A.

Proposition 4.1 *Let Π^{jit} be a JIT-OPRF protocol for a JIT family of functions f. Then given an adversary A_{prf} playing the game $\mathbf{G}^{\text{jit-prf}}_{\mathsf{f}}$ we can construct an adversary A_{jit} playing the game $\mathbf{G}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi^{\text{jit}}}$ such that*

$$\mathbf{Adv}^{\text{jit-prf}}_{\mathsf{f}}(\text{A}_{\text{prf}}) \le \mathbf{Adv}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi^{\text{jit}}}(\text{A}_{\text{jit}}) \,. \tag{1}$$

We will have $\text{Q}^{\text{TR}}(\text{A}_{\text{jit}}) = \text{Q}^{\text{GenKey}}(\text{A}_{\text{prf}})$ and $\text{Q}^{\text{CH}}(\text{A}_{\text{jit}}) = \text{Q}^{\text{Fn}}(\text{A}_{\text{prf}})$. The running time of A_{jit} is about that of A_{prf}.

5 PSI from JIT-OPRFs

One objective in the introduction of JIT-OPRFs was to better and more modularly capture the working of efficient PSI protocols. To show how this works, we present in this Section two transforms which construct PSI protocols from JIT-OPRFs. The first uses any JIT-OPRF. More formally, it is a transform $\Pi^{\mathrm{psi}} = \mathbf{J2P1}[\mathsf{f}, \Pi^{\mathrm{jit}}]$ which constructs a PSI protocol Π^{psi} from JIT function family f and its corresponding JIT-OPRF protocol Π^{jit}. The construction works for any value of the capacity cp of f, but the capacity affects the cost of Π^{psi}. Namely, the computation time of the server, and the size of its communication to the client, scale as $\mathsf{nk}(\mathsf{cp}, s_1) \cdot s_2$ where s_1, s_2 are the sizes of the client and server input sets, respectively. This cost is $s_1 \cdot s_2$ when $\mathsf{cp} = 1$ and s_2 when $\mathsf{cp} = \infty$. So $\mathsf{cp} = \infty$ results in lower server cost.

However, several techniques, such as OT extension as used in [35], result in what, in our framework, is a unit capacity JIT function family. In the PSI literature, the cost of the PSI protocol is then reduced using Cuckoo hashing. We lift this to the JIT-OPRF level to get lower-cost PSI from unit-capacity JIT-OPRFs. First, we abstract Cuckoo hashing and related techniques (there are several variants) through a definition that we give of a bin allocation scheme. Then we give a second transform that constructs a PSI protocol $\Pi^{\mathrm{psi}} = \mathbf{J2P2}[\mathsf{f}, \Pi^{\mathrm{jit}}, \mathsf{BS}]$ from a unit-capacity JIT function family f, a JIT-OPRF Π^{jit} for f, and a bin allocation scheme BS. The server cost here is $c \cdot s_2$ where c is a small constant depending on the bin allocation scheme. This does not match the cost of our first construction with an infinite capacity JIT-OPRF, but it significantly improves that construction for unit capacity.

Our proofs show security of the PSI protocols assuming security of the JIT-OPRF. While prior analyses of PSI protocols consider only a single protocol execution, our definitions allow many. We allow input sets for the parties to be adaptively selected by the adversary, as a function of random oracle queries of the latter. This is (implicitly) not allowed in some prior works. Our proofs are all tight, and in particular security does not degrade by a factor of the number of protocol executions as would happen with a hybrid argument. The results are all stated fully concretely.

This section motivates the design of JIT-ORPFs. In the next Sections, we will create JIT-OPRFs from OT-extension and VOLE . By plugging these JIT-OPRFs into our JIT-OPRF-based PSI protocols, we obtain PSI protocols. Some of these recover ones already in the literature, with a more modular treatment and a complete, concrete-security proof.

5.1 PSI from any JIT-OPRF

We give our PSI protocol $\Pi^{\mathrm{psi}} = \mathbf{J2P1}[\mathsf{f}, \Pi^{\mathrm{jit}}]$, based on a JIT-OPRF protocol Π^{jit} for a JIT family of functions f, in Figure 6. The oracle space OS for Π^{psi} is that same as that of f and Π^{jit}, and H denotes a random oracle from this space that Π^{psi} provides to Π^{jit} and algorithms of f when it runs them. Party 1 turns its input set S_1 into a vector $\mathbf{x}$ of size $s_1 = |S_1|$ via $\mathbf{x} \leftarrow \mathsf{S2V}(S_1)$. The elements of $\mathbf{x}$ are used as input to the JIT-OPRF protocol Π^{jit}, with s_1 being the input for party 2. From this party 1 gets $y_1, \ldots, y_{s_1}$ as output and party 2 gets the keys $K_1, \ldots, K_n$ with $n = \mathsf{f.nk}(\mathsf{cp}, s_1)$. Party 2 constructs a set Z by evaluating f on all the elements in its input set S_2 under all its keys, $K_1, \ldots, K_n$, and sends it to party 1. Party 1 finds the common elements between Z and $y_1, \ldots, y_{s_1}$ to construct the intersection I, that it outputs.

The following theorem shows that JIT-OPRF security of Π^{jit} implies InI security for the PSI protocol $\Pi = \Pi^{\mathrm{psi}}$ shown in Figure 6. Specifically, if Π^{jit} is JIT-OPRF1 secure then Π is InI secure for party 1 (client) and if Π^{jit} is JIT-OPRF2 secure then Π is InI secure for party 2 (server). The theorem also shows that Π is correct as long as the underlying JIT family of functions f is a JIT-PRF. (It may seem like JIT-PRF security of f is an additional requirement, but it isn't because

PSI Protocol $\mathbf{J2P1}[\mathsf{f}, \Pi^{\mathrm{jit}}]$

Based on: JIT PRF f with capacity cp, and protocol Π^{jit} for $\mathsf{F}_{\mathsf{f}}^{\mathrm{jit\text{-}oprf}}$
Party 1 input: (S_1, s_2) where $S_1 \subseteq \{0,1\}^*$ has size $s_1 = |S_1|$, and $s_2 = |S_2|$.
Party 2 input: (S_2, s_1) where $S_2 \subseteq \{0,1\}^*$ has size $s_2 = |S_2|$, and $s_1 = |S_1|$.

1. Party 1 creates a vector from its input S_1 as $\mathbf{x} \leftarrow \mathsf{S2V}(S_1)$.
2. The parties compute $((y_1, \ldots, y_{s_1}), (K_1, \ldots, K_n)) \leftarrow\!\!\$\ \mathsf{F}_{\mathsf{f}}^{\mathrm{jit\text{-}oprf}}[\mathsf{H}](\mathbf{x}, s_1)$ by running Π^{jit} where $n = \mathsf{f.nk}(\mathsf{cp}, s_1)$.
3. Party 2 creates set $Z \leftarrow \{\mathsf{f.Ev}[\mathsf{H}](K, x) \mid K \in \{K_1, \ldots, K_n\}$ and $x \in S_2\}$ and sends Z to party 1.
4. Party 1 constructs its intersection set $I \leftarrow \{\, \mathbf{x}[i] : i \in [1..s_1]$ and $y_i \in Z \,\}$.
5. Party 1 outputs I and party 2 outputs ε.

Figure 6: PSI protocol $\Pi = \mathbf{J2P1}[\mathsf{f}, \Pi^{\mathrm{jit}}]$ built from JIT-OPRF protocol Π^{jit} for a JIT family of functions f.

Proposition 4.1 says that JIT-OPRF2 security of Π^{jit} (for f) implies JIT-PRF security of f.) The results are fully concrete, and the bounds are all tight. The proof is given in Appendix B.

For an adversary $\mathrm{A}_{\mathrm{psi}}$ playing any of the games associated to the PSI protocol Π, it is useful below to let $\mathrm{SS}_{i,j}(\mathrm{A}_{\mathrm{psi}})$ denote an upper bound on the size of the set(s) of party $j \in \{1,2\}$ in the i-th query to RUN.

Theorem 5.1 *Let $\mathsf{F} = \mathsf{F}^{\mathrm{psi}}$ be the PSI functionality. Let Π^{jit} be a protocol for $\mathsf{F}_{\mathsf{f}}^{\mathrm{jit\text{-}oprf}}$ where $\mathsf{f.Ev}\colon [\mathsf{OS}] \times \mathsf{Keys} \times \{0,1\}^* \to \mathsf{Rng}$. Let $\Pi = \mathbf{J2P1}[\mathsf{f}, \Pi^{\mathrm{jit}}]$ be the PSI protocol shown in Figure 6. For brevity, below, let $q = \mathrm{Q}^{\mathrm{RUN}}(\mathrm{A}_{\mathrm{psi}})$ and $s_{i,j} = \mathrm{SS}_{i,j}(\mathrm{A}_{\mathrm{psi}})$ for $j \in \{1,2\}$ and $i \in [1..q]$. Then:*

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

$$\mathbf{Adv}^{\mathrm{corr}}_{\mathsf{F},\Pi}(\mathrm{A}_{\mathrm{psi}}) \le \mathbf{Adv}^{\mathrm{jit\text{-}prf}}_{\mathsf{f}}(\mathrm{A}_{\mathrm{prf}}) + \sum_{i=1}^{q} \frac{n_i \cdot s_{i,1} \cdot s_{i,2}}{|\mathsf{Rng}|} \ . \tag{2}$$

   Here $n_i = \mathsf{f.nk}(\mathsf{f.cp}, s_{i,1})$ for $i \in [1..q]$. We will have $\mathrm{Q}^{\mathrm{GenKey}}(\mathrm{A}_{\mathrm{prf}}) = q$ and $\mathrm{Q}^{\mathrm{Fn}}(\mathrm{A}_{\mathrm{prf}}) \le \sum_{i=1}^{q}(s_{i,1} + n_i \cdot s_{i,2})$ and $\mathrm{Q}^{\mathrm{RO}}(\mathrm{A}_{\mathrm{psi}}) = \mathrm{Q}^{\mathrm{RO}}(\mathrm{A}_{\mathrm{prf}})$. The running time of $\mathrm{A}_{\mathrm{prf}}$ is about that of $\mathrm{A}_{\mathrm{psi}}$.

2. *Π provides InI client security if Π^{jit} is JIT-OPRF1 secure: Let $\mathrm{A}_{\mathrm{psi}}$ be an adversary playing game $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F},\Pi,1}$. Then we can construct an adversary $\mathrm{A}_{\mathrm{jit}}$ such that*

$$\mathbf{Adv}^{\mathrm{ini}}_{\mathsf{F},\Pi,1}(\mathrm{A}_{\mathrm{psi}}) \le \mathbf{Adv}^{\mathrm{jit\text{-}oprf\text{-}1}}_{\mathsf{f},\Pi^{\mathrm{jit}}}(\mathrm{A}_{\mathrm{jit}}) \ . \tag{3}$$

   We will have $\mathrm{Q}^{\mathrm{RUN}}(\mathrm{A}_{\mathrm{jit}}) = q$ and $\mathrm{Q}^{\mathrm{RO}}(\mathrm{A}_{\mathrm{jit}}) = \mathrm{Q}^{\mathrm{RO}}(\mathrm{A}_{\mathrm{psi}})$. The running time of $\mathrm{A}_{\mathrm{jit}}$ is about that of $\mathrm{A}_{\mathrm{psi}}$.

3. *Π provides InI server security if Π^{jit} is JIT-OPRF2 secure: Let $\mathrm{A}_{\mathrm{psi}}$ be an adversary playing game $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F},\Pi,2}$. Then we can construct an adversary $\mathrm{A}_{\mathrm{jit}}$ such that*

$$\mathbf{Adv}^{\mathrm{ini}}_{\mathsf{F},\Pi,2}(\mathrm{A}_{\mathrm{psi}}) \le 2 \cdot \mathbf{Adv}^{\mathrm{jit\text{-}oprf\text{-}2}}_{\mathsf{f},\Pi^{\mathrm{jit}}}(\mathrm{A}_{\mathrm{jit}}) \ . \tag{4}$$

   We will have $\mathrm{Q}^{\mathrm{TR}}(\mathrm{A}_{\mathrm{jit}}) = q$ and $\mathrm{Q}^{\mathrm{CH}}(\mathrm{A}_{\mathrm{jit}}) \le n \cdot s_2$ where s_2, n are upper bounds on the sum of the sizes of the sets of party 2 and number of keys given to party 2, respectively, across all RUN queries. Also $\mathrm{Q}^{\mathrm{RO}}(\mathrm{A}_{\mathrm{jit}}) = \mathrm{Q}^{\mathrm{RO}}(\mathrm{A}_{\mathrm{psi}})$. The running time of $\mathrm{A}_{\mathrm{jit}}$ is about that of $\mathrm{A}_{\mathrm{psi}}$.

5.2 Bin-allocation schemes

In Section 5.3 we present another transform which improves upon **J2P1** when the underlying JIT family of functions has unit capacity. The analogous improvement in the PSI literature was done with Cuckoo hashing, and we use the same approach. However, Cuckoo hashing has several different forms (there may be two hash functions or three, there may be a stash or not, ...) and each prior PSI protocol explicitly describes and uses some particular form. We abstract and generalize this by defining a primitive we call a *Bin Allocation* (BA) scheme. We will discuss later how to instantiate it with Cuckoo hashing. Proceeding to the definition, a bin-allocation scheme BS defines parameters and algorithms as follows:

- $(bp, nb, ts) \leftarrow\!\!\$\ \mathsf{Pg}[\mathsf{G}](n)$— The parameter generation algorithm takes an integer $n \geq 1$ (in PSI, the size of the set of party 1) and, with access to a random oracle $\mathsf{G} \in \mathsf{OS}$, generates parameters bp, along with numbers $nb \in \mathbb{N}$ (the number of bins), and $ts \in \mathbb{N}$ (the size of the test set). While the algorithm is randomized, it is required that nb and ts are deterministic functions of n alone.
- $(\pi, D, d) \leftarrow \mathsf{Alloc}[\mathsf{G}](bp, nb, ts, S_1)$— The deterministic allocation algorithm takes a size n set $S_1 \subseteq \{0,1\}^*$. It returns a size $nb-n$ set $D \subseteq \{0,1\}^*$ of "dummy items" that satisfies $D \cap S_1 = \emptyset$, a bijection $\pi\colon S_1 \cup D \to [1..nb]$ and a boolean $d \in \{0,1\}$ that indicates success ($d = 1$) or failure ($d = 0$). Intuitively, item $x \in S_1 \cup D$ is being placed in bin number $\pi(x)$, at least as long as $d = 1$. We refer to π as the placement function.
- $T \leftarrow \mathsf{Find}[\mathsf{G}](bp, nb, ts, x)$— The deterministic bin-finding algorithm takes a point $x \in \{0,1\}^*$ and returns a set $T \subseteq [1..nb]$ of bin numbers that has size $|T| = ts$.

The description of BS consists of the algorithms $\mathsf{Pg}, \mathsf{Alloc}, \mathsf{Find}$ and also, as usual, the oracle space OS from which G is drawn.

Correctness. As a correctness condition, we require the following. Suppose that $(\pi, D, d) \leftarrow \mathsf{Alloc}[\mathsf{G}](bp, nb, ts, S_1)$ and $d = 1$, meaning allocation is successful. Then for any $x \in S_1$, if we let $T \leftarrow \mathsf{Find}[\mathsf{G}](bp, nb, ts, x)$, it will be the case that $\pi(x) \in T$. That is, the set T contains the index (number) of the bin in which x was placed. The correctness error of scheme BS is defined using the game $\mathbf{G}^{\text{corr-ba}}_{\mathsf{BS}}$ shown in Fig 7. The advantage of an adversary $\mathrm{A}_{\mathrm{cba}}$ is defined as $\mathbf{Adv}^{\text{corr-ba}}_{\mathsf{BS}}(\mathrm{A}_{\mathrm{cba}}) = \Pr[\mathbf{G}^{\text{corr-ba}}_{\mathsf{BS}}(\mathrm{A}_{\mathrm{cba}})]$. The adversary has access to a random oracle RO as well as an Alloc oracle which takes a set S as input. It wins by querying Alloc with a set S that leads to allocation failure ($d = 0$). Intuitively, correctness means the advantage of an adversary making a practical number of oracle queries is low. As usual, in our concrete framework, this will manifest as $\mathbf{Adv}^{\text{corr-ba}}_{\mathsf{BS}}(\mathrm{A}_{\mathrm{cba}})$ arising in bounds with the resources of $\mathrm{A}_{\mathrm{cba}}$ explicitly noted.

Bin allocation from cuckoo hashing. We can build a bin allocation scheme BS using Cuckoo hashing. The basic idea is that bp is a (description of) some number k (say $k = 3$) of hash functions $h_1, \ldots, h_k\colon \{0,1\}^* \to [1..nb]$. Then Alloc uses Cuckoo hashing with a stash to construct the placement function for a set S_1 of size n, and $nb - n$ will be the size of the stash. The error arises if, after a certain number of tries, Cuckoo hashing fails.

However, there are various subtleties and details to attend to. The hash functions, in Cuckoo hashing and PSI literature, are assumed to be random. This effectively means they have to be implemented using the random oracle G, which is why we define bin allocation schemes in the ROM. The first thought is to set $h_i = \mathsf{G}(i, \cdot)$. However, Cuckoo hashing and PSI literature also assume the hash functions are chosen independently of the points (elements of S_1) being hashed, and the error probability is bounded (shown low) only under this assumption. In our (realistic) setting where S_1 will be chosen by an adversary in a PSI protocol, this independence will not hold for h_i as just defined because the adversary can query the random oracle before deciding what to

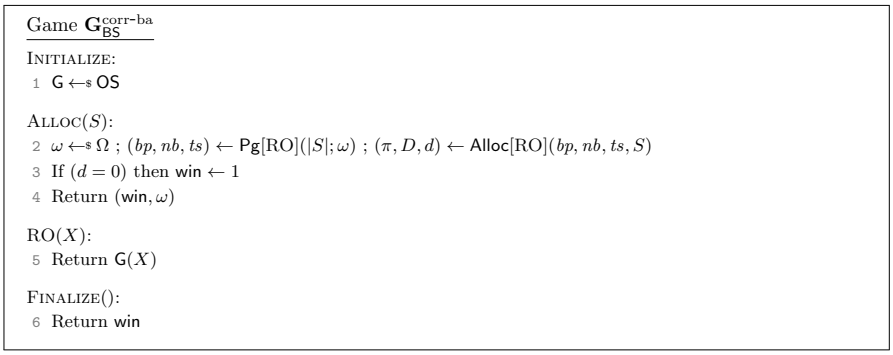


Figure 7: Game for defining correctness error of bin allocation scheme BS.

use as S_1, and thereby induce allocation failure with a high probability. The solution we suggest is salting. We have Pg pick distinct, random strings $\sigma_1, \ldots, \sigma_k$, called salts, of some (sufficiently large, say 256 bits) salt-length s, and set $bp = (\sigma_1, \ldots, \sigma_k)$. Then we define $h_i = \mathsf{G}(\sigma_i, \cdot)$ for $i \in [1..k]$. Now, as per game $\mathbf{G}^{\text{corr-ba}}_{\mathsf{BS}}$, the choice of these salts is made by Pg in response to an $\textsc{Alloc}(S_1)$ query and thus *after* the adversary has chosen S_1. Since it is unlikely to have made any queries of the form $\mathsf{G}(\sigma_i, \cdot)$ prior to providing S_1, the hash functions are independent of the inputs to which they are applied, putting us back in the usual setting.

One should show this works by giving a bound on $\mathbf{Adv}^{\text{corr-ba}}_{\mathsf{BS}}(\mathrm{A}_{\mathrm{cba}})$, as a function of the number $\mathrm{Q}^{\mathrm{RO}}(\mathrm{A}_{\mathrm{cba}})$ of queries adversary $\mathrm{A}_{\mathrm{cba}}$ makes to its random oracle, the number $\mathrm{Q}^{\textsc{Alloc}}(\mathrm{A}_{\mathrm{cba}})$ of queries it makes to its $\textsc{Alloc}$ oracle, and the salt length s. This will add a term roughly of the form $\mathrm{Q}^{\mathrm{RO}}(\mathrm{A}_{\mathrm{cba}}) \cdot \mathrm{Q}^{\textsc{Alloc}}(\mathrm{A}_{\mathrm{cba}}) \cdot k \cdot 2^{-s}$ to the classical Cuckoo hashing error. However, the latter is hard to bound. Here one has to turn to works that have attempted this, either analytically [45, 61] or experimentally [52].

5.3 PSI from JIT-OPRFs and Bin Allocation

Our second transform **J2P2** constructs the PSI protocol $\Pi^{\text{psi}} = \mathbf{J2P2}[\mathsf{f}, \Pi^{\text{jit}}, \mathsf{BS}]$, shown in Figure 8, from a bin allocation scheme BS and a JIT-OPRF protocol Π^{jit} associated to a unit capacity JIT family of functions f. We see it as using random oracles G and H, the former for BS and the latter for $\mathsf{f}, \Pi^{\text{jit}}$. (More formally, let OS_1 be the oracle space of BS and OS_2 that of f and Π^{jit}. Then the oracle space OS of Π^{psi} can be seen as consisting of functions J where $\mathsf{J}(\mathtt{g}, \cdot) \in \mathsf{OS}_1$ and $\mathsf{J}(\mathtt{h}, \cdot) \in \mathsf{OS}_2$, where $\mathtt{g}, \mathtt{h}$ are distinct "name" strings, and we are letting $\mathsf{G} = \mathsf{J}(\mathtt{g}, \cdot)$ and $\mathsf{H} = \mathsf{J}(\mathtt{h}, \cdot)$ for simplicity. Note that these two functions must be independently selected.) The objective in using BS is to reduce the size of the set Z that party 2 sends by limiting the number of keys under which $\mathsf{f.Ev}$ is evaluated for each element in its input set. Towards this, party 1 starts by running parameter generation for BS to get (bp, nb, ts) and allocates its input set S_1 as $(\pi, D, d) \leftarrow \mathsf{Alloc}[\mathsf{G}](bp, nb, ts, S_1)$. If this fails ($d = 0$), it sends d to party 2 (to indicate abort) and rejects. (This corresponds to setting the decision in its final state to be 0.) Else, it sends (bp, nb, ts) to party 2. Note that since π is a bijection we have $|S_1 \cup D| = nb$. Party 1 constructs nb-vector $\mathbf{x}$ via $\mathbf{x}[\pi(x)] \leftarrow x$ for all $x \in S_1 \cup D$. The parties then execute Π^{jit} with $\mathbf{x}$ being the input for party 1 and nb the input for party 2. Party 2 gets the keys $(K_1, \ldots, K_{nb})$ and party 1 gets values $y_1, \ldots, y_{nb}$ such that $y_i = \mathsf{f.Ev}[\mathsf{H}](K_i, \mathbf{x}[i])$. Party 2 now constructs its set Z, for elements $x \in S_2$, by finding the bins for x using $\mathsf{BS.Find}$ and evaluating x on the keys indexed by these bins. This is the improvement over **J2P1** where party 2 now needs to evaluate x under ts keys rather than all keys as in **J2P1**. Finally, party 1 computes its output as before, using $y_1, \ldots, y_{nb}$ and Z, while making sure dummy elements are not added to the intersection output.

PSI Protocol $\mathbf{J2P2}[\mathsf{f}, \Pi^{\mathrm{jit}}, \mathsf{BS}]$

Based on: Bin Allocation scheme BS, JIT PRF f with $\mathsf{f.cp} = 1$, and protocol Π^{jit} for $\mathsf{F}_{\mathsf{f}}^{\text{jit-oprf}}$.
Party 1 input: (S_1, s_2) where $S_1 \subseteq U$ has size $s_1 = |S_1|$, and $s_2 = |S_2|$.
Party 2 input: (S_2, s_1) where $S_2 \subseteq U$ has size $s_2 = |S_2|$, and $s_1 = |S_1|$.

1. Party 1 picks parameters $(bp, nb, ts) \leftarrow\!\!\$\ \mathsf{BS.Pg}[\mathsf{G}](|S_1|)$ and allocates the elements of S_1 via $(\pi, D, d) \leftarrow \mathsf{Alloc}[\mathsf{G}](bp, nb, ts, S_1)$. If $d = 1$ then it sends (bp, nb, ts) to party 2 and goes to Step **2** below. Else it sends d to party 2 (as an indication to abort), rejects (formally, sets $st_1.\mathrm{dec} = 0$ and $st_1.\mathrm{out} = \perp$, where st_1 is its state) and terminates.
2. Party 1 constructs its input $\mathbf{x} = (\mathbf{x}[1], \ldots, \mathbf{x}[nb])$ for the JIT-OPRF protocol Π^{jit} via: For $x \in S_1 \cup D$ do $i \leftarrow \pi(x)$; $\mathbf{x}[i] \leftarrow x$. Note that $|S_1 \cup D| = nb$.
3. The parties compute $((y_1, \ldots, y_{nb}), (K_1, \ldots, K_{nb})) \leftarrow\!\!\$\ \mathsf{F}_{\mathsf{f}}^{\text{jit-oprf}}[\mathsf{H}](\mathbf{x}, nb)$ by running Π^{jit}.
4. Party 2 constructs set Z as follows: For $x \in S_2$ do $T \leftarrow \mathsf{Find}[\mathsf{G}](bp, nb, ts, x)$; $P \leftarrow \{K_i : i \in T\}$; $Z \leftarrow Z \cup \{\mathsf{f.Ev}[\mathsf{H}](K, x) \mid K \in P\}$. It then sends Z to party 1.
5. Party 1 constructs its intersection set $I \leftarrow \{\mathbf{x}[i] : i \in [1..nb] \text{ and } y_i \in Z \text{ and } \mathbf{x}[i] \in S_1\}$.
6. Party 1 outputs I and party 2 outputs ε.

Figure 8: PSI protocol $\Pi = \mathbf{J2P2}[\mathsf{f}, \Pi^{\mathrm{jit}}, \mathsf{BS}]$ built from JIT-OPRF protocol Π^{jit} for a JIT family of functions f, and bin allocation scheme BS.

The following theorem gives results regarding correctness and InI security of the PSI protocol $\Pi^{\mathrm{psi}} = \mathbf{J2P2}[\mathsf{f}, \Pi^{\mathrm{jit}}, \mathsf{BS}]$. As for Theorem 5.1, the results are fully concrete. A new element is the error probability of the bin allocation scheme. It affects correctness of Π^{psi}, as expected, but note that it also affects the bound in the case of client security for Π^{psi}. The bounds are tight. The proof is given in Appendix C .

As above, for an adversary $\mathrm{A}_{\mathrm{psi}}$ playing any of the games associated to the PSI protocol Π, it is useful to let $\mathrm{SS}_{i,j}(\mathrm{A}_{\mathrm{psi}})$ denote an upper bound on the size of the set(s) of party $j \in \{1, 2\}$ in the i-th query to RUN. We also let $\mathrm{TS}_i(\mathrm{A}_{\mathrm{psi}})$ denote the value of ts in the output of the execution of $\mathsf{BS.Pg}$ by party 1 in the i-th call to RUN.

Theorem 5.2 *Let $\mathsf{F} = \mathsf{F}^{\mathrm{psi}}$ be the PSI functionality. Let Π^{jit} be a protocol for $\mathsf{F}_{\mathsf{f}}^{\text{jit-oprf}}$ based on f with $\mathsf{f.Ev}\colon [\mathsf{OS}_2] \times \mathsf{Keys} \times \{0,1\}^* \to \mathsf{Rng}$. Let BS be a bin allocation scheme. Let $\Pi = \mathbf{J2P2}[\mathsf{f}, \Pi^{\mathrm{jit}}, \mathsf{BS}]$ be the PSI protocol as in Figure 8. For brevity, below, let $q = \mathrm{Q}^{\mathrm{RUN}}(\mathrm{A}_{\mathrm{psi}})$ and $s_{i,j} = \mathrm{SS}_{i,j}(\mathrm{A}_{\mathrm{psi}})$ and $ts_i = \mathrm{TS}_i(\mathrm{A}_{\mathrm{psi}})$ for $j \in \{1, 2\}$ and $i \in [1..q]$. Then:*

1. *Π is correct if f is a JIT-PRF: Let $\mathrm{A}_{\mathrm{psi}}$ be an adversary playing game $\mathbf{G}^{\mathrm{corr}}_{\mathsf{F},\Pi}$. Then we can construct adversaries $\mathrm{A}_{\mathrm{prf}}$ and $\mathrm{A}_{\mathrm{cba}}$ such that*

$$\mathbf{Adv}^{\mathrm{corr}}_{\mathsf{F},\Pi}(\mathrm{A}_{\mathrm{psi}}) \le \mathbf{Adv}^{\text{jit-prf}}_{\mathsf{f}}(\mathrm{A}_{\mathrm{prf}}) + \mathbf{Adv}^{\text{corr-ba}}_{\mathsf{BS}}(\mathrm{A}_{\mathrm{cba}}) + \sum_{i=1}^{q} \frac{ts_i \cdot s_{i,1} \cdot s_{i,2}}{|\mathsf{Rng}|} \ , \tag{5}$$

   We will have $\mathrm{Q}^{\mathrm{GENKEY}}(\mathrm{A}_{\mathrm{prf}}) = \mathrm{Q}^{\mathrm{ALLOC}}(\mathrm{A}_{\mathrm{cba}}) = q$ and $\mathrm{Q}^{\mathrm{FN}}(\mathrm{A}_{\mathrm{prf}}) \le \sum_{i=1}^{q}(s_{i,1} + ts_i \cdot s_{i,2})$ and $\mathrm{Q}^{\mathrm{RO}}(\mathrm{A}_{\mathrm{prf}}) \le \mathrm{Q}^{\mathrm{RO}}(\mathrm{A}_{\mathrm{psi}})$ and $\mathrm{Q}^{\mathrm{RO}}(\mathrm{A}_{\mathrm{cba}}) \le \mathrm{Q}^{\mathrm{RO}}(\mathrm{A}_{\mathrm{psi}})$. The running times of $\mathrm{A}_{\mathrm{prf}}$ and $\mathrm{A}_{\mathrm{cba}}$ are about that of $\mathrm{A}_{\mathrm{psi}}$.
2. *Π provides InI client security if Π^{jit} is JIT-OPRF1 secure: Let $\mathrm{A}_{\mathrm{psi}}$ be an adversary playing game $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F},\Pi,1}$. Then we can construct adversaries $\mathrm{A}_{\mathrm{jit}}$ and $\mathrm{A}_{\mathrm{cba}}$ such that*

$$\mathbf{Adv}^{\mathrm{ini}}_{\mathsf{F},\Pi,1}(\mathrm{A}_{\mathrm{psi}}) \le \mathbf{Adv}^{\text{jit-oprf-1}}_{\mathsf{f},\Pi^{\mathrm{jit}}}(\mathrm{A}_{\mathrm{jit}}) + 2 \cdot \mathbf{Adv}^{\text{corr-ba}}_{\mathsf{BS}}(\mathrm{A}_{\mathrm{cba}}) \ . \tag{6}$$

   We will have $\mathrm{Q}^{\mathrm{CH}}(\mathrm{A}_{\mathrm{jit}}) = q$ and $\mathrm{Q}^{\mathrm{ALLOC}}(\mathrm{A}_{\mathrm{cba}}) = 2q$ and $\mathrm{Q}^{\mathrm{RO}}(\mathrm{A}_{\mathrm{jit}}) \le \mathrm{Q}^{\mathrm{RO}}(\mathrm{A}_{\mathrm{psi}})$ and $\mathrm{Q}^{\mathrm{RO}}(\mathrm{A}_{\mathrm{cba}}) \le \mathrm{Q}^{\mathrm{RO}}(\mathrm{A}_{\mathrm{psi}})$. The running times of $\mathrm{A}_{\mathrm{jit}}$ and $\mathrm{A}_{\mathrm{cba}}$ are about that of $\mathrm{A}_{\mathrm{psi}}$.
3. *Π provides InI server security if Π^{jit} is JIT-OPRF2 secure: Let $\mathrm{A}_{\mathrm{psi}}$ be an adversary playing*

game $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F},\Pi,2}$. *Then we can construct an adversary* $\mathrm{A}_{\mathrm{jit}}$ *such that*

$$\mathbf{Adv}^{\mathrm{ini}}_{\mathsf{F},\Pi,2}(\mathrm{A}_{\mathrm{psi}}) \le 2 \cdot \mathbf{Adv}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi^{\mathrm{jit}}}(\mathrm{A}_{\mathrm{jit}}) \ . \tag{7}$$

We will have $\mathrm{Q}^{\mathrm{TR}}(\mathrm{A}_{\mathrm{jit}}) = \mathrm{Q}^{\mathrm{RUN}}(\mathrm{A}_{\mathrm{psi}})$ *and* $\mathrm{Q}^{\mathrm{CH}}(\mathrm{A}_{\mathrm{jit}}) \le t \cdot s_2$ *and* $\mathrm{Q}^{\mathrm{RO}}(\mathrm{A}_{\mathrm{jit}}) \le \mathrm{Q}^{\mathrm{RO}}(\mathrm{A}_{\mathrm{psi}})$, *where* $s_2 = s_{1,2} + \cdots + s_{q,2}$ *and* $t = ts_1 + \cdots + ts_q$. *The running time of* $\mathrm{A}_{\mathrm{jit}}$ *is about that of* $\mathrm{A}_{\mathrm{psi}}$.

6 JIT-OPRF from OT

In this section we use the OT-extension technique of IKNP [30], as used for BaRK-OPRF and PSI in KKRT [35], to build a JIT-OPRF from OT. Theorem 6.1 proves both JIT-OPRF1 and JIT-OPRF2 security of this construction, with explicit bounds. Combining this with Theorem 5.2, we get a full proof and concrete-security bounds for the KKRT PSI protocol.

6.1 Construction

PRELIMINARIES. We will be considering matrices over the field GF(2) with the latter identified with $\{0,1\}$. Note the GF(2)-field operations of "+" and "·" are the same as XOR and AND, respectively, over $\{0,1\}$. We will use notation for matrices that was given in Section 2.

OBLIVIOUS TRANSFER. An execution of the basic 1-out-of-2 OT functionality is written as $(\varepsilon, \mathbf{q}) \leftarrow \mathsf{F}^{\mathrm{ot}}((\mathbf{t}_0, \mathbf{t}_1), (n, b))$. Here party 1, the client, is the "sender." It has as input a pair $\mathbf{t}_0, \mathbf{t}_1 \in \{0,1\}^{n\times 1}$ of column vectors over a set $\{0,1\}$. Party 2, the server, is the "receiver." It has n and a selection bit $b \in \{0,1\}$. The functionality provides the receiver with $\mathbf{q} = \mathbf{t}_b$ while the sender gets nothing (formally the empty string). It is convenient to extend this to $(\varepsilon, \mathbf{Q}) \leftarrow \mathsf{F}^{\mathrm{ot}}_k((\mathbf{T}_0, \mathbf{T}_1), (n, s))$ for $k \ge 1$, defined as

$\mathsf{F}^{\mathrm{ot}}_k((\mathbf{T}_0, \mathbf{T}_1), (n, s))$ // $\mathbf{T}_0, \mathbf{T}_1 \in \{0,1\}^{n\times k}$ and $s \in \{0,1\}^k$
For $j = 1, \ldots, k$ do $(\varepsilon, \mathbf{Q}[\cdot][j]) \leftarrow \mathsf{F}^{\mathrm{ot}}((\mathbf{T}_0[\cdot][j], \mathbf{T}_1[\cdot][j]), (n, s[j]))$
Return $(\varepsilon, \mathbf{Q})$ // $\mathbf{Q} \in \{0,1\}^{n\times k}$

This results in $\mathbf{Q}$ having, as its j-th column, the j-th column of $\mathbf{T}_{s[j]}$, meaning $\mathbf{Q}[\cdot][j] = \mathbf{T}_{s[j]}[\cdot][j]$ for $j \in [1..k]$. Note this can be written as

$$\mathbf{Q} = \mathbf{T}_0 + (\mathbf{T}_1 - \mathbf{T}_0) \cdot \mathbf{diag}(s) \ , \tag{8}$$

which we will use later. We will assume given a protocol Π^{ot} for the basic functionality F^{ot}. We assume it has perfect correctness. We assume it does not use a random oracle. (The extension to the case that it does is easy but one must take care that this RO is independent of any used by the calling JIT-OPRF protocol, which can be ensured with domain separation [4].) Our JIT-OPRF protocol will use it to implement the $\mathsf{F}^{\mathrm{ot}}_k$ functionality as above, using Π^{ot} in place of F^{ot}. Here k will be a statistical security parameter of the JIT-OPRF protocol.

Since the F^{ot} functionality is deterministic, we can use, for Π^{ot}, the generic InI security definitions of BRRA [7], recalled in Section 3. Many efficient OT protocols exist [5, 42, 15].

CONSTRUCTIONS. We define first a JIT family of functions f and then a corresponding JIT-OPRF Π. Formally, each is built via a transform.

Transform **F1** constructs a JIT family of functions $\mathsf{f} = \mathbf{F1}[k, \ell]$, where ℓ will be the output length of the function. It will be unit capacity, meaning have capacity $\mathsf{cp} = 1$. It uses RO H where $\mathsf{H}(1,\cdot)\colon \{0,1\}^* \to \{0,1\}^{1\times k}$ (the output is a row-vector of length k) and $\mathsf{H}(2,\cdot)\colon (\{0,1\}^* \times \{0,1\}^{1\times k})$

$\underline{\mathbf{F1}[k,\ell].\mathsf{Ev}[\mathsf{H}](K,x)}$: // $x \in \{0,1\}^*$

1 $(\mathbf{q}, s) \leftarrow K$ // $(\mathbf{q}, s) \in \{0,1\}^{1\times k} \times \{0,1\}^k$
2 $\mathbf{y} \leftarrow \mathbf{q} + \mathsf{H}(1,x) \cdot \mathbf{diag}(s)$ // $\mathbf{y} \in \{0,1\}^{1\times k}$
3 $Y \leftarrow \mathsf{H}(2,(x,\mathbf{y}))$ // $Y \in \{0,1\}^\ell$
4 Return Y

$\underline{\mathbf{F1}[k,\ell].\mathsf{Kg}[\mathsf{H}](x_1,\ldots,x_n)}$: // $x_1,\ldots,x_n \in \{0,1\}^*$

1 $s \leftarrow\!\!\$\ \{0,1\}^k$; $\mathbf{T} \leftarrow\!\!\$\ \{0,1\}^{n\times k}$
2 For $i = 1,\ldots,n$ do
3 $\quad \mathbf{h}[i][\cdot] \leftarrow \mathsf{H}(1,x_i)$ // $\mathbf{h} \in \{0,1\}^{n\times k}$
4 $\mathbf{Q} \leftarrow \mathbf{T} + \mathbf{h} \cdot \mathbf{diag}(s)$ // $\mathbf{Q} \in \{0,1\}^{n\times k}$
5 For $i = 1,\ldots,n$ do $K_i \leftarrow (\mathbf{Q}[i][\cdot], s)$
6 Return $(K_1,\ldots,K_n)$

JIT-OPRF Protocol $\Pi = \mathbf{J1}[k,\ell,\Pi^{\mathrm{ot}}]$ for realizing functionality $\mathsf{F}_{\mathsf{f}}^{\text{jit-oprf}}$, where $\mathsf{f} = \mathbf{F1}[k,\ell]$

Based on: Protocol Π^{ot} for F^{ot}, statistical security parameter k, and output length ℓ
Party 1 input: $x_1,\ldots,x_n \in \{0,1\}^*$.
Party 2 input: n

1. Party 1 defines $\mathbf{h} \in \{0,1\}^{n\times k}$ by $\mathbf{h}[i][\cdot] \leftarrow \mathsf{H}(1,x_i)$ for $i \in [1..n]$. It picks $\mathbf{T}_0 \leftarrow\!\!\$\ \{0,1\}^{n\times k}$ and defines $\mathbf{T}_1 \leftarrow \mathbf{T}_0 + \mathbf{h}$.
2. Party 2 picks $s \leftarrow\!\!\$\ \{0,1\}^k$.
3. The parties compute $(\varepsilon, \mathbf{Q}) \leftarrow \mathsf{F}_k^{\mathrm{ot}}((\mathbf{T}_0,\mathbf{T}_1),(n,s))$ by running the given Π^{ot} OT protocol k times, their inputs in the j-th run being $(\mathbf{T}_0[\cdot][j], \mathbf{T}_1[\cdot][j])$ and $(n, s[j])$, respectively, for $j \in [1..k]$. Note by Eq. (8) we have $\mathbf{Q} = \mathbf{T}_0 + \mathbf{h} \cdot \mathbf{diag}(s)$.
4. Party 1 outputs $Y_1,\ldots,Y_n$ defined by $Y_i \leftarrow \mathsf{H}(2,(x_i,\mathbf{T}_0[i][\cdot]))$ for $i \in [1..n]$.
5. Party 2 outputs $(K_1,\ldots,K_n)$ defined by $K_i \leftarrow (\mathbf{Q}[i][\cdot], s)$ for $i \in [1..n]$.

Figure 9: Top: JIT family of functions $\mathsf{f} = \mathbf{F1}[k,\ell]$. Bottom: JIT-OPRF Protocol $\Pi = \mathbf{J1}[k,\ell,\Pi^{\mathrm{ot}}]$ for realizing functionality $\mathsf{F}_{\mathsf{f}}^{\text{jit-oprf}}$ based on OT extension. The random oracles here are $\mathsf{H}(1,\cdot)\colon \{0,1\}^* \to \{0,1\}^{1\times k}$ and $\mathsf{H}(2,\cdot)\colon (\{0,1\}^* \times \{0,1\}^{1\times k}) \to \{0,1\}^\ell$.

$\to \{0,1\}^\ell$, so that OS is formally the set of all such functions H. It has $\mathsf{Dom} = \{0,1\}^*$ and $\mathsf{Rng} = \{0,1\}^\ell$. It has $\mathsf{Keys} = \{0,1\}^{1\times k} \times \{0,1\}^k$, meaning a key $K = (\mathbf{q}, s)$ is a pair of k-vectors over $\{0,1\}$, with the first regarded as a row vector. The evaluation and key-generation algorithms are shown at the top of Figure 9. The output of key-generation is a list of n keys since the capacity is 1.

Transform **J1** constructs a JIT-OPRF protocol $\Pi = \mathbf{J1}[k,\ell,\Pi^{\mathrm{ot}}]$ where Π^{ot} is a protocol for F^{ot}. It has the same random oracles as f. The protocol is shown at the bottom of Figure 9. It uses the OT-extension technique of IKNP [30] as used for BaRK-OPRF and PSI in KKRT [35]. We are adapting this to build a JIT-PRF.

CORRECTNESS OF **J1**. Consider the protocol at the bottom of Figure 9. Suppose party 1 has input $x_1,\ldots,x_n \in \{0,1\}^*$ and party 2 has input n. Suppose party 1 outputs $Y_1,\ldots,Y_n$ at step **4** and party 2 outputs $(K_1,\ldots,K_n)$ at step **5**. For correctness of the JIT-OPRF $\Pi = \mathbf{J1}[k,\ell,\Pi^{\mathrm{ot}}]$, we need to verify that $Y_i = \mathbf{F1}[k,\ell].\mathsf{Ev}[\mathsf{H}](K_i,x_i)$ for all $i \in [1..n]$.

Let $i \in [1..n]$. Now we have:

$$\begin{aligned}
\mathbf{T}_0[i][\cdot] &= \mathbf{T}_0[i][\cdot] + \mathsf{H}(1,x_i) \cdot \mathbf{diag}(s) + \mathsf{H}(1,x_i) \cdot \mathbf{diag}(s) \\
&= \mathbf{T}_0[i][\cdot] + \mathbf{h}[i][\cdot] \cdot \mathbf{diag}(s) + \mathsf{H}(1,x_i) \cdot \mathbf{diag}(s) \qquad \text{// because } \mathbf{h}[i][\cdot] = \mathsf{H}(1,x_i) \text{ by step } \mathbf{1} \\
&= \mathbf{Q}[i][\cdot] + \mathsf{H}(1,x_i) \cdot \mathbf{diag}(s) \qquad \text{// because } \mathbf{Q} = \mathbf{T}_0 + \mathbf{h} \cdot \mathbf{diag}(s) \text{ by Eq. (8)} .
\end{aligned}$$

Now from step **4** we have $Y_i = \mathsf{H}(2,(x_i,\mathbf{T}_0[i][\cdot]))$. From step **5** we know that $K_i = (\mathbf{Q}[i][\cdot], s)$. From the top left panel of Figure 9 we can now conclude that $Y_i = \mathbf{F1}[k,\ell].\mathsf{Ev}[\mathsf{H}](K_i,x_i)$ as required.

COMPARISONS. KKRT [35] use a Pseudorandom Code (PRC). Their definition of a PRC asks that, for any distinct x, x', if we pick a random code C, then $\mathsf{C}(x) + \mathsf{C}(x')$ is unlikely to have small Hamming Weight. Here, however, x, x' cannot depend on the code. Our JIT-OPRF1 and JIT-OPRF2 definitions allow adaptive and dynamic choices of inputs and queries, under which x, x' could depend on the code, and thus a PRC as defined in KKRT no longer suffices. In its place, we use a RO, namely $\mathsf{H}(1,\cdot)$. The proof of part (2) of Theorem 6.1 does have to argue a similar

Hamming Weight condition. This proof ends up being quite involved.

6.2 Security result

The following implies that $\Pi = \mathbf{J1}[k, \ell, \Pi^{\mathrm{ot}}]$ is JIT-OPRF1-secure if Π^{ot} is InI secure for the client (party 1, sender), and JIT-OPRF2-secure if Π^{ot} is InI secure for the server (party 2, receiver). After stating the Theorem, we note a generalization, and then discuss the bounds and their implications for parameter choices and efficiency. We note that the bounds below correct and update the ones from the preliminary version of this paper [6]. Following that we proceed to the proof, giving this separately for the two parts of the Theorem..

Theorem 6.1 *Let $\mathsf{f} = \mathbf{F1}[k, \ell]$ be the JIT family of functions shown in Figure 9. Let Π^{ot} be a protocol for F^{ot}. Let $\Pi = \mathbf{J1}[k, \ell, \Pi^{\mathrm{ot}}]$ be the JIT-OPRF protocol shown in Figure 9. Then:*

1. *Π provides JIT-OPRF1 security if Π^{ot} is InI secure for the client (sender): Let $\mathrm{A}_{\mathrm{jit}}$ be an adversary playing game $\mathbf{G}^{\text{jit-oprf-1}}_{\mathsf{f},\Pi}$. Then we can construct an adversary A_{ot} playing game $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F}^{\mathrm{ot}},\Pi^{\mathrm{ot}},1}$ such that*

$$\mathbf{Adv}^{\text{jit-oprf-1}}_{\mathsf{f},\Pi}(\mathrm{A}_{\mathrm{jit}}) \le \mathbf{Adv}^{\mathrm{ini}}_{\mathsf{F}^{\mathrm{ot}},\Pi^{\mathrm{ot}},1}(\mathrm{A}_{\mathrm{ot}}) \ . \tag{9}$$

   We will have $\mathrm{Q}^{\mathrm{Run}}(\mathrm{A}_{\mathrm{ot}}) = k \cdot \mathrm{Q}^{\mathrm{CH}}(\mathrm{A}_{\mathrm{jit}})$. The running time of A_{ot} is about the same as that of $\mathrm{A}_{\mathrm{jit}}$.
2. *Π provides JIT-OPRF2 security if Π^{ot} is InI secure for the server (receiver): Let $\mathrm{A}_{\mathrm{jit}}$ be an adversary playing game $\mathbf{G}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi}$. Let $Q_i = \mathrm{Q}^{\mathrm{RO}(i,\cdot)}(\mathrm{A}_{\mathrm{jit}})$ ($i \in \{1, 2\}$). Assume the total number of inputs across all TR queries of $\mathrm{A}_{\mathrm{jit}}$ is L. Let $Q_c = \mathrm{Q}^{\mathrm{CH}}(\mathrm{A}_{\mathrm{jit}})$. Then we can construct an adversary A_{ot} playing game $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F}^{\mathrm{ot}},\Pi^{\mathrm{ot}},2}$ such that*

$$\mathbf{Adv}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi}(\mathrm{A}_{\mathrm{jit}}) \le \mathbf{Adv}^{\mathrm{ini}}_{\mathsf{F}^{\mathrm{ot}},\Pi^{\mathrm{ot}},2}(\mathrm{A}_{\mathrm{ot}}) + \delta \ , \tag{10}$$

   where

$$\delta = \frac{L \cdot (Q_c + Q_2)}{2^{k/5}} + \frac{Q_1^2}{2 \cdot 2^{k/4}} + \frac{L^2}{2^{k+1}} \ . \tag{11}$$

   We will have $\mathrm{Q}^{\mathrm{Run}}(\mathrm{A}_{\mathrm{ot}}) = k \cdot \mathrm{Q}^{\mathrm{TR}}(\mathrm{A}_{\mathrm{jit}})$. The running time of A_{ot} is about the same as that of $\mathrm{A}_{\mathrm{jit}}$.

NOTE. Eq. (11) is updated from the proceedings version of this paper [6]. In particular, there is a factor of L multiplied to the first term which was not present before.

GENERALIZATION. The bound of Eq. (11) can be generalized as follows. Let α be any parameter in the range $0 < \alpha < 1/2$ and let $\beta = (1 - 2\alpha)^2 \log_2(e)/2$. Then in Eq. (11), we can replace $k/5$ by αk and $k/4$ by βk. One can then pick α depending on Q_1, Q_2, Q_c, L to minimize δ. The generalization arises from replacing the use of Lemma 6.4 with Lemma 6.3 in the proof of part (2) of Theorem 6.1 in Section 6.4.

DISCUSSION OF BOUNDS IN THEOREM. The proof of part (1) is tight. In particular, the bound of Eq. (9) does not degrade multiplicatively with the number of TR queries of $\mathrm{A}_{\mathrm{jit}}$, as would arise from a hybrid argument. Instead, this is reflected in the number of RUN queries of A_{ot}. This motivates asking how the security of OT protocols such as those of [5, 42, 15] degrades with the number of RUN queries, meaning the number of invocations of the protocol. This has not been studied, leading to the interesting open question of whether one can give a tight reduction of the

InI security of these OT protocols to the underlying hard algebraic problem. We note that the concrete security of the OT protocol of [15] has been studied in [22], but under a simulation-based definition and for a single protocol execution.

The Eq. (10) bound for part (2) has a computational term, which is the advantage of an OT adversary, and a statistical term δ. The bound is tight in the computational term, and all the discussion above applies. The statistical term $\delta = \delta_1 + \delta_2 + \delta_3$ is the sum of three terms of the form $\delta_1 = P_1 \cdot 2^{-k/5}$ and $\delta_2 = P_2 \cdot 2^{-k/4}$ and $\delta_3 = P_3 \cdot 2^{-k}$, where P_1, P_2, P_3 can be determined from Eq. (11). As the statistical security parameter k increases, δ goes down and so security goes up, but the cost of the OT goes up too, so we want to pick k as small as we can subject to a desired upper bound on δ. In this regard, the first good news is that $\delta_1, \delta_2, \delta_3$ are exponentially decreasing in k. The second good news is that the polynomial terms P_1, P_2, P_3 are small. (Quadratic terms are the norm in bounds due to collisions, and there is nothing larger.) The bad news is that the exponents of δ_1, δ_2 multiply k by factors, namely $1/5$ and $1/4$, that are small. These elements arise from Lemma 6.4, which is used to ensure that various sums in the proof of Section 6.4 have high Hamming weight, and the fact that these sums are adaptively and dynamically determined by oracle queries of $\mathrm{A}_{\mathrm{jit}}$. For Eq. (11), we have deliberately made proof choices that ensure that the exponent $k/4$ for δ_2 is larger than the $k/5$ for δ_1, meaning the exponential degradation is faster for the former, to compensate for P_2 being larger than P_1.

As an example, ball-park estimate, say $Q_c = L = 2^{20} \approx 10^6$. These parameters reflect usage in an application like PSI, and hence may be assumed not too large. The random oracle queries however represent off-line computation of a hash function by the adversary, and so it behoves us to set them quite large, say $Q_1 = Q_2 = 2^{60}$. Now say our target is to ensure $\delta \leq 2^{-32}$. We want to pick k as small as possible to allow this. We can neglect δ_3 since the exponent k is very large compared to $k/5$ and $k/4$. Then it suffices that $\delta_1, \delta_2 \leq 2^{-33}$. We have $\delta_1 \approx 2^{80} \cdot 2^{-k/5}$ and this is at most 2^{-33} for $k \geq 565$. We have $\delta_2 = 2^{119} \cdot 2^{-k/4}$ and this is at most 2^{-33} for $k = 152 \cdot 4 = 608$, so we conclude with the choice $k = 608$.

We note that the generalization of Eq. (11) discussed after the statement of Theorem 6.1 can be used to reduce the value of k through a choice of α depending on the Q_1, Q_2, Q_c, L in this example. Specifically, setting $\alpha \approx 0.197$ results in the same bound of 2^{-32} on δ but with $k = 574$.

6.3 Proof of Theorem 6.1 Part (1)

Proof of Theorem 6.1, Part (1): Adversary A_{ot} runs $c' \leftarrow\!\!\$\ \mathrm{A}_{\mathrm{jit}}[\mathrm{CH}, \mathrm{RO}]$. It sets $b' \leftarrow c'$, and returns b' as its own output. Here CH, RO are subroutines that A_{ot} defines to replace the oracles of the same names that $\mathrm{A}_{\mathrm{jit}}$ expects. In defining the first, A_{ot} uses its own RUN oracle, and also its RO subroutine, as follows:

$\underline{\mathrm{CH}((x_{1,0}, \ldots, x_{n,0}), (x_{1,1}, \ldots, x_{n,1}))}$:

For $i = 1, \ldots, n$ do $\mathbf{h}_0[i][\cdot] \leftarrow \mathrm{RO}(1, x_{i,0})$; $\mathbf{h}_1[i][\cdot] \leftarrow \mathrm{RO}(1, x_{i,1})$
$s \leftarrow\!\!\$\ \{0,1\}^k$; $\mathbf{T}_{0,0} \leftarrow\!\!\$\ \{0,1\}^{n \times k}$; $\mathbf{T}_{1,0} \leftarrow \mathbf{T}_{0,0} + \mathbf{h}_0$
$\mathbf{T}_{0,1} \leftarrow \mathbf{T}_{0,0} + \mathbf{h}_0 \cdot \mathbf{diag}(s) - \mathbf{h}_1 \cdot \mathbf{diag}(s)$
$\mathbf{T}_{1,1} \leftarrow \mathbf{T}_{0,1} + \mathbf{h}_1$
For $i = 1, \ldots, k$ do
  $(\boldsymbol{\tau}[i], \boldsymbol{\omega}[i]) \leftarrow \mathrm{RUN}((\mathbf{T}_{0,0}[\cdot][i], \mathbf{T}_{1,0}[\cdot][i]), (\mathbf{T}_{0,1}[\cdot][i], \mathbf{T}_{1,1}[\cdot][i]), (n, s[i]))$
$(\varepsilon, \mathbf{Q}) \leftarrow \mathsf{F}_k^{\mathrm{ot}}((\mathbf{T}_{0,0}, \mathbf{T}_{1,0}), (n, s))$ // output same as $\mathsf{F}_k^{\mathrm{ot}}((\mathbf{T}_{0,1}, \mathbf{T}_{1,1}), (n, s))$
For $i = 1, \ldots, n$ do $K_i \leftarrow (\mathbf{Q}[i][\cdot], s)$
Return $((K_1, \ldots, K_n), \boldsymbol{\tau}, (\boldsymbol{\omega}, s))$

A_{ot} defines subroutine RO directly using lazy sampling as follows:

$\underline{\mathrm{RO}(a, X)}$:

10 If $\mathrm{R}[a, X] \neq \perp$ then return $\mathrm{R}[a, X]$
11 If $a = 1$ then $\mathrm{R}[a, X] \leftarrow\!\!\$\ \{0,1\}^k$; Return $\mathrm{R}[a, X]$
12 $\mathrm{R}[a, X] \leftarrow\!\!\$\ \{0,1\}^\ell$; Return $\mathrm{R}[a, X]$

In CH subroutine, adversary A_{ot} sees two input tuples for party 1 in f. It follows the protocol Π to construct inputs for its k RUN oracle queries as Π^{ot} is run k times in Π. Finally, it uses the output of its RUN queries to construct and return the keys, transcript and coins for $\mathrm{A}_{\mathrm{jit}}$. As a result, the challenge bit for A_{ot} now becomes the challenge bit for $\mathrm{A}_{\mathrm{jit}}$ and we get the claimed result in Eq. (9).

Note that RUN oracle queries, in InI, require that the two pairs of input, that make up the query, result in the same functionality output for the adversary, which here would be the columns of $\mathbf{Q}$. Therefore, the RUN oracle queries need to be carefully constructed. In its response to a $\mathrm{CH}(X_0, X_1)$ query, X_0, X_1 being the input tuples as shown above, A_{ot} first follows Π to compute matrices $\mathbf{h}_0$ and $\mathbf{h}_1$, corresponding to X_0 and X_1 respectively, and picks $s, \mathbf{T}_{0,0}$. It then computes $\mathbf{T}_{1,0}$ as in Π. This completes the required computation for X_0. Now, A_{ot} needs to construct matrices $\mathbf{T}_{1,0}$ and $\mathbf{T}_{1,1}$ for X_1 such that $\mathbf{T}_{s[i],0}[\cdot][i] = \mathbf{T}_{s[i],1}[\cdot][i]$ for all $i \in [1..k]$ to satisfy the requirement in RUN oracle. This is because the i-th RUN oracle query would need to be $((\mathbf{T}_{0,0}[\cdot][i], \mathbf{T}_{1,0}[\cdot][i]), (\mathbf{T}_{0,1}[\cdot][i], \mathbf{T}_{1,1}[\cdot][i]), (n, s[i]))$. Note that by Eq. (8), the output after all executions of Π^{ot} on columns of $\mathbf{T}_{0,0}, \mathbf{T}_{1,0}$ on bits in s would be $\mathbf{T}_{0,0} + \mathbf{h}_0 \cdot \mathbf{diag}(s)$. This means, if $\mathbf{T}_{0,1}$ is constructed such that $\mathbf{T}_{0,1} + \mathbf{h}_1 \cdot \mathbf{diag}(s) = \mathbf{T}_{0,0} + \mathbf{h}_0 \cdot \mathbf{diag}(s)$, then all the k RUN oracle queries would satisfy the required condition mentioned above. This is exactly how $\mathbf{T}_{0,1}$ is computed in line 3 in the response above which also ensures $\mathbf{Q}$ can be computed as in line 7. Note that since $\mathbf{T}_{0,0}$ is uniformly chosen, $\mathbf{T}_{0,1}$ is a uniform matrix from $\{0,1\}^{n \times k}$ as well. Therefore, the above satisfies all the requirements needed to correctly simulate the CH oracle for $\mathrm{A}_{\mathrm{jit}}$ and we get the claimed result in Eq. (9) as explained before. ∎

6.4 Proof of Theorem 6.1 Part (2)

Towards the proof of part (2) of Theorem 6.1 we start with some Lemmas. The following is a special case of Hoeffding's Tail Inequality [25].

Lemma 6.2 (Hoeffding Inequality **[25]**) *Let $\mathsf{X}_1, \ldots, \mathsf{X}_k$ be independent random variables taking values in $\{0,1\}$. Let $\mathsf{X} = \mathsf{X}_1 + \cdots + \mathsf{X}_k$ and $\mu = \mathbf{E}[\mathsf{X}]$. Let $t \geq 0$. Then*

$$\Pr[\mathsf{X} - \mu \geq kt] \leq e^{-2kt^2} .$$

Recall that the Hamming Weight $\mathbf{w}_{\mathrm{H}}(a)$ of a string (binary vector) a is the number of 1s in a.

Lemma 6.3 *Suppose $0 < \alpha < 1/2$ and let $\beta = (1-2\alpha)^2 \log_2(e)/2$. Suppose $a \leftarrow\!\!\$\ \{0,1\}^k$. Then $\Pr[\mathbf{w}_{\mathrm{H}}(a) < \alpha k] \leq 2^{-\beta k}$.*

Proof of Lemma 6.3: For $i \in [1..k]$ let random variable X_i take value 1 if $a[i] = 0$ and 0 otherwise. Let $\mathsf{X} = \mathsf{X}_1 + \cdots + \mathsf{X}_k$. Let $\mu = \mathbf{E}[\mathsf{X}] = \mathbf{E}[\mathsf{X}_1] + \cdots + \mathbf{E}[\mathsf{X}_k] = k \cdot (1/2) = k/2$. Then

$$\begin{aligned}\Pr[\mathbf{w}_{\mathrm{H}}(a) < \alpha k] &= \Pr[\mathsf{X} \geq (1-\alpha)k] = \Pr[\mathsf{X} - \mu \geq (1/2 - \alpha)k] \\ &\leq e^{-2k(1/2-\alpha)^2} = e^{-k(1-2\alpha)^2/2} = 2^{-k(1-2\alpha)^2 \log_2(e)/2} = 2^{-\beta k} ,\end{aligned}$$

Games G_0, G_1

$\mathrm{TR}(x_1, \ldots, x_n)$:
1 $r \leftarrow r+1$; $n_r \leftarrow n$
2 For $i = 1, \ldots, n$ do $\mathbf{h}_r[i][\cdot] \leftarrow \mathrm{RO}(1, x_i)$
3 $\mathbf{T}_{r,0} \leftarrow\!\!\$\ \{0,1\}^{n \times k}$; $\mathbf{T}_{r,1} \leftarrow \mathbf{T}_{r,0} + \mathbf{h}_r$; $s_{r,0}, s_{r,1} \leftarrow\!\!\$\ \{0,1\}^k$
4 For $j = 1, \ldots, k$ do
5 $\quad \boldsymbol{\omega}_1[j], \boldsymbol{\omega}_2[j] \leftarrow\!\!\$\ \Omega$
6 $\quad (\boldsymbol{\tau}[j], st_1, st_2) \leftarrow \mathbf{Int}_{\Pi^{\mathrm{ot}}}((\mathbf{T}_{r,0}[\cdot][j], \mathbf{T}_{r,1}[\cdot][j]), (n_r, s_{r,0}[j]); \boldsymbol{\omega}_1[j], \boldsymbol{\omega}_2[j])$ // Game G_0 only
7 $\quad (\boldsymbol{\tau}[j], st_1, st_2) \leftarrow \mathbf{Int}_{\Pi^{\mathrm{ot}}}((\mathbf{T}_{r,0}[\cdot][j], \mathbf{T}_{r,1}[\cdot][j]), (n_r, s_{r,1}[j]); \boldsymbol{\omega}_1[j], \boldsymbol{\omega}_2[j])$ // Game G_1 only
8 $\mathbf{Q}_r \leftarrow \mathbf{T}_{r,0} + \mathbf{h}_r \cdot \mathbf{diag}(s_{r,1})$
9 For $i = 1, \ldots, n$ do $\mathbf{Y}_r[i] \leftarrow \mathrm{RO}(2, (x_i, \mathbf{T}_{r,0}[i][\cdot]))$; $\mathrm{T}[r, i, x_i] \leftarrow \mathbf{Y}_r[i]$
10 Return $(\mathbf{Y}, \boldsymbol{\tau}, (\mathbf{T}_{r,0}, \boldsymbol{\omega}_1))$

$\mathrm{CH}(r', i, x)$:
11 If not $(r' \le r$ and $i \le n_{r'})$ then return $\perp$
12 If $\mathrm{T}[r', i, x] \neq \perp$ then return $\mathrm{T}[r', i, x]$
13 $\mathbf{y} \leftarrow \mathbf{Q}_{r'}[i][\cdot] + \mathrm{RO}(1, x) \cdot \mathbf{diag}(s_{r',1})$; $\mathrm{T}[r', i, x] \leftarrow \mathrm{RO}(2, (x, \mathbf{y}))$
14 Return $\mathrm{T}[r', i, x]$

$\mathrm{RO}(a, X)$:
15 If $\mathrm{R}[a, X] \neq \perp$ then return $\mathrm{R}[a, X]$
16 If $a = 1$ then $\mathrm{R}[a, X] \leftarrow\!\!\$\ \{0,1\}^k$; Return $\mathrm{R}[a, X]$
17 $\mathrm{R}[a, X] \leftarrow\!\!\$\ \{0,1\}^\ell$; Return $\mathrm{R}[a, X]$

FINALIZE(c'):
18 Return $[[c' = 1]]$

Figure 10: Games for the proof of part (2) of Theorem 6.1.

where the first inequality above uses Lemma 6.2 with $t = 1/2 - \alpha$. ▌

As a special case of the above, we obtain the following.

Lemma 6.4 *Suppose* $a \leftarrow\!\!\$\ \{0,1\}^k$. *Then* $\Pr[\mathbf{w}_{\mathrm{H}}(a) < k/5] < 2^{-k/4}$.

Proof of Lemma 6.4: Apply Lemma 6.3 with $\alpha = 1/5$. Then $\beta = (1-2\alpha)^2 \log_2(e)/2 = (9/50)\log_2(e) > 1/4$ so $2^{-\beta k} < 2^{-k/4}$. ▌

Lemma 6.5 *Suppose* $a \in \{0,1\}^{1 \times k}$ *and* $s \leftarrow\!\!\$\ \{0,1\}^k$. *Let* B *be an adversary. Then* $\Pr[\mathrm{B}(a) = a \cdot \mathbf{diag}(s)] \le 2^{-\mathbf{w}_{\mathrm{H}}(a)}$.

Proof of Lemma 6.5: Let $\mathbf{t} = a \cdot \mathbf{diag}(s)$. If $a[i] = 0$ then B knows that $\mathbf{t}[i] = 0$. If $a[i] = 1$ then $\mathbf{t}[i] = s[i]$ so B can predict it with probability at most $1/2$. So

$$\Pr[\mathrm{B}(a) = \mathbf{t}] \le \prod_{i\,:\,a[i]=1} \frac{1}{2} = 2^{-\mathbf{w}_{\mathrm{H}}(a)}$$

as required. ▌

Proof of Theorem 6.1, Part (2): We use a sequence of games $\mathrm{G}_0, \mathrm{G}_1, \mathrm{G}_2, \mathrm{G}_3, \mathrm{G}_4$ that will be executed with $\mathrm{A}_{\mathrm{jit}}$. For brevity it is useful to let

$$\epsilon_i = \Pr[\mathrm{G}_i(\mathrm{A}_{\mathrm{jit}})] \ .$$

Games $\mathrm{G}_0, \mathrm{G}_1$ of Figure 10 start from game $\mathbf{G}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi}$ of Figure 5. They then detail the choices and steps as per the descriptions of f and Π in Figure 9. Instead of just one, two server secrets $s_{r,0}, s_{r,1}$ are chosen at line 3. The games are the same except that line 6 is only in G_0 and line 7 is only in

G_1. The difference between these two lines is only in the input for party 2 in the call for Π^{ot}, this input being $s_{r,0}$ for G_0 and $s_{r,1}$ for G_1. Note that other than at these lines, both games consistently use $s_{r,1}$ as the server secret. The result is that G_1 is equivalent to the $c = 1$ (i.e., "real") case of $\mathbf{G}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi}$. We now write

$$\epsilon_1 = (\epsilon_1 - \epsilon_0) + \epsilon_0 \ .$$

We build adversary A_{ot} such that

$$\epsilon_1 - \epsilon_0 \leq \mathbf{Adv}^{\text{ini}}_{\mathsf{F}^{\mathrm{ot}},\Pi^{\mathrm{ot}},2}(\mathrm{A}_{\mathrm{ot}}) \ . \tag{12}$$

Adversary A_{ot} runs $c' \leftarrow\!\!\$\ \mathrm{A}_{\mathrm{jit}}[\mathrm{TR}, \mathrm{CH}, \mathrm{RO}]$. It sets $b' \leftarrow c'$, and returns b' as its own output. Here $\mathrm{TR}, \mathrm{CH}, \mathrm{RO}$ are subroutines that A_{ot} defines to replace the oracles of the same names that $\mathrm{A}_{\mathrm{jit}}$ expects. In defining the first, A_{ot} uses its own RUN oracle, as follows:

$\underline{\mathrm{TR}(x_1, \ldots, x_n)}$:

1 $r \leftarrow r + 1$; $n_r \leftarrow n$
2 For $i = 1, \ldots, n$ do $\mathbf{h}_r[i][\cdot] \leftarrow \mathrm{RO}(1, x_i)$
3 $\mathbf{T}_{r,0} \leftarrow\!\!\$\ \{0,1\}^{n \times k}$; $\mathbf{T}_{r,1} \leftarrow \mathbf{T}_{r,0} + \mathbf{h}_r$; $s_{r,0}, s_{r,1} \leftarrow\!\!\$\ \{0,1\}^k$
4 For $j = 1, \ldots, k$ do
5 $\quad (\boldsymbol{\tau}[j], \boldsymbol{\omega}_1[j]) \leftarrow \textsc{Run}((n_r, s_{r,0}), (n_r, s_{r,1}), (\mathbf{T}_{r,0}[\cdot][j], \mathbf{T}_{r,1}[\cdot][j]))$
6 $\mathbf{Q}_r \leftarrow \mathbf{T}_{r,0} + \mathbf{h}_r \cdot \mathbf{diag}(s_{r,1})$
7 For $i = 1, \ldots, n$ do $\mathbf{Y}_r[i] \leftarrow \mathrm{RO}(2, (x_i, \mathbf{T}_{r,0}[i][\cdot]))$; $\mathrm{T}[r, i, x_i] \leftarrow \mathbf{Y}_r[i]$
8 Return $(\mathbf{Y}, \boldsymbol{\tau}, (\mathbf{T}_{r,0}, \boldsymbol{\omega}_1))$

Subroutines CH, RO are defined exactly as in games $\mathrm{G}_0, \mathrm{G}_1$, and we do not repeat the code here. (Note that A_{ot} does not have a random oracle. Its calls to RO, above and in its CH subroutine, refer to calls to the RO subroutine it itself defines.) Let b be the challenge bit in game $\mathbf{G}^{\text{ini}}_{\mathsf{F}^{\mathrm{ot}},\Pi^{\mathrm{ot}},1}$. Then, in the execution of $\mathrm{A}_{\mathrm{jit}}$ by A_{ot}, the former is seeing the same oracle responses that it would get from G_b. Hence

$$\begin{aligned}
\mathbf{Adv}^{\text{ini}}_{\mathsf{F}^{\mathrm{ot}},\Pi^{\mathrm{ot}},2}(\mathrm{A}_{\mathrm{ot}}) &= \Pr\left[\ b' = 1 \ \mid\ b = 1\ \right] - \Pr\left[\ b' = 1 \ \mid\ b = 0\ \right] \\
&= \Pr\left[\ c' = 1 \ \mid\ b = 1\ \right] - \Pr\left[\ c' = 1 \ \mid\ b = 0\ \right] \\
&= \Pr[\mathrm{G}_1(\mathrm{A}_{\mathrm{jit}})] - \Pr[\mathrm{G}_0(\mathrm{A}_{\mathrm{jit}})] = \epsilon_1 - \epsilon_0 \ ,
\end{aligned}$$

which justifies Eq. (12).

The rest of the proof is devoted to bounding ϵ_0. This is the more difficult part. As a start, let us ask what we have gained by moving from G_1 to G_0. The answer is that in G_0, the OT, at line 6, uses a secret $s_{r,0}$ that is different from the $s_{r,1}$ used in the rest of the game. This means that the OT will not leak any information to $\mathrm{A}_{\mathrm{jit}}$ about $s_{r,1}$, so $\mathrm{A}_{\mathrm{jit}}$ has a priori no information about $s_{r,1}$. This means that the remaining argument is information-theoretic.

Consider games $\mathrm{G}_2, \mathrm{G}_3$ in Figure 11, where G_2 includes the boxed code and G_3 does not. For brevity of the code of these games, we have defined

$$\begin{aligned}
\Delta(u, v, x) &= \mathbf{Q}_u[v][\cdot] + \mathrm{RO}(1, x) \cdot \mathbf{diag}(s_{u,1}) \\
&= \mathbf{T}_{u,0}[v][\cdot] + (\mathbf{h}_u[v][\cdot] + \mathrm{RO}(1, x)) \cdot \mathbf{diag}(s_{u,1}) \ .
\end{aligned}$$

This is a possible value of $\mathbf{y}$ such that $\mathrm{RO}(2, (x, \mathbf{y}))$ is an output of f. The games start from G_0

Games $\boxed{\mathrm{G}_2}$, G_3

$\mathrm{TR}(x_1, \ldots, x_n)$:
1 $r \leftarrow r+1$; $n_r \leftarrow n$
2 For $i = 1, \ldots, n$ do $\mathbf{h}_r[i][\cdot] \leftarrow \mathrm{RO}(1, x_i)$
3 $\mathbf{T}_{r,0} \leftarrow\!\!\$\ \{0,1\}^{n \times k}$; $\mathbf{T}_{r,1} \leftarrow \mathbf{T}_{r,0} + \mathbf{h}_r$; $s_{r,0}, s_{r,1} \leftarrow\!\!\$\ \{0,1\}^k$
4 For $j = 1, \ldots, k$ do
5 $\quad \boldsymbol{\omega}_1[j], \boldsymbol{\omega}_2[j] \leftarrow\!\!\$\ \Omega$
6 $\quad (\boldsymbol{\tau}[j], st_1, st_2) \leftarrow \mathbf{Int}_{\Pi^{\mathrm{ot}}}((\mathbf{T}_{r,0}[\cdot][j], \mathbf{T}_{r,1}[\cdot][j]), (n_r, s_{r,0}[j]); \boldsymbol{\omega}_1[j], \boldsymbol{\omega}_2[j])$
7 $\mathbf{Q}_r \leftarrow \mathbf{T}_{r,0} + \mathbf{h}_r \cdot \mathbf{diag}(s_{r,1})$
8 For $i = 1, \ldots, n$ do $\mathbf{Y}_r[i] \leftarrow \mathrm{RO}(2, (x_i, \mathbf{T}_{r,0}[i][\cdot]))$; $\mathrm{T}[r, i, x_i] \leftarrow \mathbf{Y}_r[i]$
9 Return $(\mathbf{Y}, \boldsymbol{\tau}, (\mathbf{T}_{r,0}, \boldsymbol{\omega}_1))$

$\mathrm{CH}(r', i, x)$:
10 If not $(r' \le r$ and $i \le n_{r'})$ then return $\perp$
11 If $\mathrm{T}[r', i, x] \neq \perp$ then return $\mathrm{T}[r', i, x]$
12 $\mathbf{y} \leftarrow \Delta(r', i, x)$
13 $\mathrm{T}[r', i, x] \leftarrow\!\!\$\ \{0,1\}^\ell$
14 If $(\mathrm{R}[2, (x, \mathbf{y})] \neq \perp)$ then $\mathsf{bad} \leftarrow 1$; $\boxed{\mathrm{T}[r', i, x] \leftarrow \mathrm{R}[2, (x, \mathbf{y})]}$
15 If $(\exists\, (u, v)\ :\ (u, v) \neq (r', i)$ and $\mathrm{T}[u, v, x] \neq \perp$ and $\mathbf{y} = \Delta(u, v, x))$ then
16 $\quad \mathsf{bad} \leftarrow 1$; $\boxed{\mathrm{T}[r', i, x] \leftarrow \mathrm{T}[u, v, x]}$
17 Return $\mathrm{T}[r', i, x]$

$\mathrm{RO}(a, X)$:
18 If $\mathrm{R}[a, X] \neq \perp$ then return $\mathrm{R}[a, X]$
19 If $a = 1$ then $\mathrm{R}[a, X] \leftarrow\!\!\$\ \{0,1\}^k$; Return $\mathrm{R}[a, X]$
20 $(x, \mathbf{y}) \leftarrow X$; $\mathrm{R}[a, X] \leftarrow\!\!\$\ \{0,1\}^\ell$
21 If $(\exists\, (u, v)\ :\ \mathrm{T}[u, v, x] \neq \perp$ and $\mathbf{y} = \Delta(u, v, x))$ then
22 $\quad \mathsf{bad} \leftarrow 1$; $\boxed{\mathrm{R}[a, X] \leftarrow \mathrm{T}[u, v, x]}$
23 Return $\mathrm{R}[a, X]$

$\textsc{Finalize}(c')$:
24 Return $[[c' = 1]]$

Figure 11: More games for the proof of part (2) of Theorem 6.1. The function Δ is defined in the text. The boxed code is included only in G_2.

and our first claim is that G_2 is equivalent to G_0, meaning

$$\epsilon_2 = \epsilon_0\ . \tag{13}$$

Let us explain the games and justify Eq. (13). Oracle TR in G_2 is the same as in G_0. In CH, the reply $\mathrm{T}[r', i, x]$ is, optimistically, chosen at random at line 13. This however could be incorrect, for several reasons. The first, tested at line 14, is that $\mathrm{RO}(2, (x, \mathbf{y}))$ has already been queried, in which case, in G_0, it would be the case that $\mathrm{T}[r', i, x]$ equals $\mathrm{R}[2, (x, \mathbf{y})]$. This is ensured in G_2 by the boxed code in line 14. The second, more subtle reason the optimistic choice of line 13 could be incorrect is that the $\mathbf{y}$ from line 12 had appeared previously in a CH or TR query, meaning it was equal to $\Delta(u, v, x)$ for some $(u, v) \neq (r', i)$. Indeed, in that case, in G_0, it would be the case that $\mathrm{T}[r', i, x]$ would equal $\mathrm{T}[u, v, x]$. (This is true even if no $\mathrm{RO}(2, \cdot)$ queries have been made so far.) Line 15 tests for such a collision. If it occurs, line 16 sets bad to true, and the boxed code then corrects the optimistic choice of $\mathrm{T}[r', i, x]$. At line 15 it could be that there are multiple pairs (u, v) satisfying the condition, so which do we pick to use in the boxed code at line 16? It does not matter, because the T-values of those choices would be the same. We now discuss RO. In the $a = 1$ case it is the same as in G_0. In the $a = 2$ case, line 20 parses X as $(x, \mathbf{y})$ and optimistically picks the reply $\mathrm{R}[a, X]$ at random. This is however incorrect if $\mathbf{y}$ already occurred as a potential input for $\mathrm{RO}[2, (x, \cdot)]$ in a CH query, meaning if the condition tested at line 21 is true, because in that case, in G_0, the values $\mathrm{R}[a, X]$ and $\mathrm{T}[u, v, x]$ would be the same. Line 22 makes the corresponding correction. Again at line 21 there could be many choices of (u, v) but due to lines 14–16, they

would yield the same result for line 22. With this we have given the core part of the explanation for Eq. (13), but we note that there is more to be checked. For example, queries to CH and RO can occur in any order, and we must check that the different corrections in the boxed code "play well" together regardless of this order. Similarly, we must check that, if the conditions at lines 14, 15 both trigger, then consistency with G_0 is still maintained.

Games $\mathrm{G}_2, \mathrm{G}_3$ are identical-until-bad as defined in [9]. By the Fundamental Lemma of Game Playing in [9] (Lemma 2.1) we thus have

$$\epsilon_2 - \epsilon_3 \leq \Pr[\mathrm{G}_3(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{bad}] \ . \tag{14}$$

Recall that G_3 does not contain the boxed code. Thus, even though various conditions are tested at lines 14, 15, 21, they do not change the optimistic choices of lines 13, 20, meaning those are what the respective oracles return. The result is that G_3 is equivalent to the $c = 0$ ("random") case of game $\mathbf{G}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi}$. Thus we have

$$\begin{aligned}
\mathbf{Adv}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi}(\mathrm{A}_{\mathrm{jit}}) &= \epsilon_1 - \epsilon_3 \\
&= (\epsilon_1 - \epsilon_0) + (\epsilon_0 - \epsilon_2) + (\epsilon_2 - \epsilon_3) \\
&\leq \mathbf{Adv}^{\text{ini}}_{\mathsf{F}^{\mathrm{ot}},\Pi^{\mathrm{ot}},1}(\mathrm{A}_{\mathrm{ot}}) + \Pr[\mathrm{G}_3(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{bad}] \ .
\end{aligned} \tag{15}$$

To bound $\Pr[\mathrm{G}_3(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{bad}]$, consider games $\mathrm{G}_4, \mathrm{G}_5$ of Figure 12, where G_5 contains the boxed code and G_4 does not. Where G_3 sets bad, games $\mathrm{G}_4, \mathrm{G}_5$ set win, and FINALIZE returns win. Lines 5,6,24,25 in G_4 can be ignored, since they do not change what the game returns. Thus we have

$$\Pr[\mathrm{G}_3(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{bad}] = \epsilon_4 \ .$$

Games $\mathrm{G}_4, \mathrm{G}_5$ are identical-until-bad as defined in [9]. By the Fundamental Lemma of Game Playing in [9] (Lemma 2.1), we thus have

$$\epsilon_4 = \epsilon_5 + (\epsilon_4 - \epsilon_5) \leq \epsilon_5 + \Pr[\mathrm{G}_5(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{bad}] \ . \tag{16}$$

We now have to bound the two terms above. We start with the second, claiming that

$$\Pr[\mathrm{G}_5(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{bad}] \leq \frac{L^2}{2^{k+1}} + \binom{Q_1}{2} \cdot \frac{1}{2^{k/4}} \ . \tag{17}$$

The two terms above correspond to the setting of bad at lines 6, 25, respectively. We justify Eq. (17) by considering each of these in turn. The probability that line 6 sets bad is $\binom{n}{2}/2^k$ so overall it contributes

$$\begin{aligned}
\frac{1}{2^k} \cdot \sum_{i=1}^{r} \binom{n_i}{2} &= \frac{1}{2^{k+1}} \cdot \sum_{i=1}^{r} n_i(n_i - 1) \leq \frac{1}{2^{k+1}} \cdot \sum_{i=1}^{r} n_i^2 \\
&\leq \frac{1}{2^{k+1}} \cdot \left(\sum_{i=1}^{r} n_i \right)^2 \leq \frac{L^2}{2^{k+1}} \ ,
\end{aligned}$$

which justifies the first term of Eq. (17). At line 25, the uniform choice of w from line 23 means that each $\mathrm{R}[1, X'] + w$ is uniformly distributed over $\{0,1\}^k$ and thus by Lemma 6.4, the probability that $\mathbf{w}_{\mathrm{H}}(\mathrm{R}[1, X'] + w) < k/5$ is at most $2^{-k/4}$. Overall, there are $\binom{Q_1}{2}$ such pairs that could result in bad being set, which justifies the second term of Eq. (17). This completes the justification of the latter equation.

Games $\boxed{\mathrm{G}_4}$, G_5

$\mathrm{TR}(x_1, \ldots, x_n)$:
1 $r \leftarrow r+1$; $n_r \leftarrow n$
2 For $i = 1, \ldots, n$ do $\mathbf{h}_r[i][\cdot] \leftarrow \mathrm{RO}(1, x_i)$; $x_{r,i} \leftarrow x_i$
3 For $i = 1, \ldots, n$ do
4 $\quad \mathbf{T}_{r,0}[i][\cdot] \leftarrow\!\!\$\ \{0,1\}^{1\times k}$
5 $\quad$ If $(\exists v < i\ :\ \mathbf{T}_{r,0}[i][\cdot] = \mathbf{T}_{r,0}[v][\cdot])$ then
6 $\quad\quad$ $\mathsf{bad} \leftarrow 1$; $\boxed{\mathbf{T}_{r,0}[i][\cdot] \leftarrow\!\!\$\ \{0,1\}^{1\times k} \setminus \{\, \mathbf{T}_{r,0}[v]\ :\ 1 \le v \le i-1 \,\}}$
7 $\mathbf{T}_{r,1} \leftarrow \mathbf{T}_{r,0} + \mathbf{h}_r$; $s_{r,0}, s_{r,1} \leftarrow\!\!\$\ \{0,1\}^k$
8 For $j = 1, \ldots, k$ do
9 $\quad \boldsymbol{\omega}_1[j], \boldsymbol{\omega}_2[j] \leftarrow\!\!\$\ \Omega$
10 $\quad (\boldsymbol{\tau}[j], st_1, st_2) \leftarrow \mathbf{Int}_{\Pi^{\mathrm{ot}}}((\mathbf{T}_{r,0}[\cdot][j], \mathbf{T}_{r,1}[\cdot][j]), (n_r, s_{r,0}[j]); \boldsymbol{\omega}_1[j], \boldsymbol{\omega}_2[j])$
11 $\mathbf{Q}_r \leftarrow \mathbf{T}_{r,0} + \mathbf{h}_r \cdot \mathbf{diag}(s_{r,1})$
12 For $i = 1, \ldots, n$ do $\mathbf{Y}_r[i] \leftarrow \mathrm{RO}(2, (x_i, \mathbf{T}_{r,0}[i][\cdot]))$; $\mathrm{T}[r, i, x_i] \leftarrow \mathbf{Y}_r[i]$
13 Return $(\mathbf{Y}, \boldsymbol{\tau}, (\mathbf{T}_{r,0}, \boldsymbol{\omega}_1))$

$\mathrm{CH}(r', i, x)$:
14 If not $(r' \le r$ and $i \le n_{r'})$ then return $\perp$
15 If $\mathrm{T}[r', i, x] \neq \perp$ then return $\mathrm{T}[r', i, x]$
16 $\mathbf{y} \leftarrow \Delta(r', i, x)$
17 $\mathrm{T}[r', i, x] \leftarrow\!\!\$\ \{0,1\}^\ell$
18 If $(\mathrm{R}[2, (x, \mathbf{y})] \neq \perp)$ then $\mathsf{win} \leftarrow 1$
19 If $(\exists\, (u, v)\ :\ (u, v) \neq (r', i)$ and $\mathrm{T}[u, v, x] \neq \perp$ and $\mathbf{y} = \Delta(u, v, x))$ then $\mathsf{win} \leftarrow 1$
20 Return $\mathrm{T}[r', i, x]$

$\mathrm{RO}(a, X)$:
21 If $\mathrm{R}[a, X] \neq \perp$ then return $\mathrm{R}[a, X]$
22 If $a = 1$ then
23 $\quad w \leftarrow\!\!\$\ \{0,1\}^k$
24 $\quad \mathrm{G} \leftarrow \{\, w' \in \{0,1\}^k\ :\ \mathbf{w}_{\mathrm{H}}(\mathrm{R}[1, X'] + w') \ge k/5$ for all X' with $\mathrm{R}[1, X'] \neq \perp \,\}$
25 $\quad$ If $(\exists\, X'\ :\ X' \neq X$ and $\mathrm{R}[1, X'] \neq \perp$ and $\mathbf{w}_{\mathrm{H}}(\mathrm{R}[1, X'] + w) < k/5)$ then $\mathsf{bad} \leftarrow 1$; $\boxed{w \leftarrow\!\!\$\ \mathrm{G}}$
26 $\quad \mathrm{R}[1, X] \leftarrow w$; Return $\mathrm{R}[1, X]$
27 $(x, \mathbf{y}) \leftarrow X$; $\mathrm{R}[a, X] \leftarrow\!\!\$\ \{0,1\}^\ell$
28 If $(\exists\, (u, v)\ :\ \mathrm{T}[u, v, x] \neq \perp$ and $\mathbf{y} = \Delta(u, v, x))$ then $\mathsf{win} \leftarrow 1$
29 Return $\mathrm{R}[a, X]$

$\textsc{Finalize}(c')$:
30 Return win

Figure 12: More games for the proof of part (2) of Theorem 6.1. The function Δ is defined in the text. The boxed code is included only in G_5.

Now we have to bound ϵ_5. Towards this, game G_6 of Figure 13 rewrites G_5 in simplified form. We have

$$\epsilon_5 = \epsilon_6\ .$$

We now bound ϵ_6. A key observation towards this is that the adversary has no information about $s_{1,1}, \ldots, s_{r,1}$. We first claim

$$\Pr[\mathrm{G}_6(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{win} \text{ at line 15}] \le Q_c \cdot 2^{-k/5} \tag{18}$$

$$\Pr[\mathrm{G}_6(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{win} \text{ at line 23}] \le L \cdot Q_2 \cdot 2^{-k/5}\ . \tag{19}$$

Observe that at line 15 we will have $x \neq x_{r',i}$ due to line 12. Due to line 21, it must be that $\mathbf{w}_{\mathrm{H}}(\mathrm{R}[1, x] + \mathrm{R}[1, x_{r',i}]) \ge k/5$. Then the probability that $\mathrm{A}_{\mathrm{jit}}$ make query $\mathsf{H}(2, (x, \Delta(r', i, x))$ is at most the probability that it computes $(\mathrm{R}[1, x] + \mathrm{R}[1, x_{r',i}]) \cdot s_{r',1}$. But above we noted that $\mathrm{A}_{\mathrm{jit}}$ has no information about $s_{r',1}$, so this probability is at most $2^{-k/5}$ by Lemma 6.5. Accounting for the Q_c queries to CH, this justifies Eq. (18). Similarly at line 23 we will have $x \neq x_{u,v}$ since $\mathrm{T}[u, v, x] \neq \perp$ as per the test on the same line, and thus the same argument holds. Accounting for

Game G_6

$\mathrm{TR}(x_1, \ldots, x_n)$:
1 $r \leftarrow r+1$; $n_r \leftarrow n$
2 For $i = 1, \ldots, n$ do $\mathbf{h}_r[i][\cdot] \leftarrow \mathrm{RO}(1, x_i)$; $x_{r,i} \leftarrow x_i$
3 For $i = 1, \ldots, n$ do $\mathbf{T}_{r,0}[i][\cdot] \leftarrow\!\!\$\ \{0,1\}^{1\times k} \setminus \{\, \mathbf{T}_{r,0}[v] \,:\, 1 \le v \le i-1 \,\}$
4 $\mathbf{T}_{r,1} \leftarrow \mathbf{T}_{r,0} + \mathbf{h}_r$; $s_{r,0}, s_{r,1} \leftarrow\!\!\$\ \{0,1\}^k$
5 For $j = 1, \ldots, k$ do
6 $\quad \boldsymbol{\omega}_1[j], \boldsymbol{\omega}_2[j] \leftarrow\!\!\$\ \Omega$
7 $\quad (\boldsymbol{\tau}[j], st_1, st_2) \leftarrow \mathbf{Int}_{\Pi^{\mathrm{ot}}}((\mathbf{T}_{r,0}[\cdot][j], \mathbf{T}_{r,1}[\cdot][j]), (n_r, s_{r,0}[j]); \boldsymbol{\omega}_1[j], \boldsymbol{\omega}_2[j])$
8 $\mathbf{Q}_r \leftarrow \mathbf{T}_{r,0} + \mathbf{h}_r \cdot \mathbf{diag}(s_{r,1})$
9 For $i = 1, \ldots, n$ do $\mathbf{Y}_r[i] \leftarrow \mathrm{RO}(2, (x_i, \mathbf{T}_{r,0}[i][\cdot]))$; $\mathrm{T}[r, i, x_i] \leftarrow \mathbf{Y}_r[i]$
10 Return $(\mathbf{Y}, \boldsymbol{\tau}, (\mathbf{T}_{r,0}, \boldsymbol{\omega}_1))$

$\mathrm{CH}(r', i, x)$:
11 If not $(r' \le r$ and $i \le n_{r'})$ then return $\perp$
12 If $\mathrm{T}[r', i, x] \ne \perp$ then return $\mathrm{T}[r', i, x]$
13 $\mathbf{y} \leftarrow \Delta(r', i, x)$
14 $\mathrm{T}[r', i, x] \leftarrow\!\!\$\ \{0,1\}^\ell$
15 If $(\mathrm{R}[2, (x, \mathbf{y})] \ne \perp)$ then $\mathsf{win} \leftarrow 1$
16 If $(\exists\, (u, v) \,:\, (u, v) \ne (r', i)$ and $\mathrm{T}[u, v, x] \ne \perp$ and $\mathbf{y} = \Delta(u, v, x))$ then $\mathsf{win} \leftarrow 1$
17 Return $\mathrm{T}[r', i, x]$

$\mathrm{RO}(a, X)$:
18 If $\mathrm{R}[a, X] \ne \perp$ then return $\mathrm{R}[a, X]$
19 If $a = 1$ then
20 $\quad \mathrm{G} \leftarrow \{\, w' \in \{0,1\}^k \,:\, \mathbf{w}_{\mathrm{H}}(\mathrm{R}[1, X'] + w') \ge k/5$ for all X' with $\mathrm{R}[1, X'] \ne \perp \,\}$
21 $\quad w \leftarrow\!\!\$\ \mathrm{G}$; $\mathrm{R}[1, X] \leftarrow w$; Return $\mathrm{R}[1, X]$
22 $(x, \mathbf{y}) \leftarrow X$; $\mathrm{R}[a, X] \leftarrow\!\!\$\ \{0,1\}^\ell$
23 If $(\exists\, (u, v) \,:\, \mathrm{T}[u, v, x] \ne \perp$ and $\mathbf{y} = \Delta(u, v, x))$ then $\mathsf{win} \leftarrow 1$
24 Return $\mathrm{R}[a, X]$

$\textsc{Finalize}(c')$:
25 Return win

Figure 13: Final game for the proof of part (2) of Theorem 6.1. The function Δ is defined in the text.

L possible (at most) values of $\Delta(u, v, x)$ in each $\mathrm{RO}(2, \cdot)$ query and Q_2 such queries, this justifies Eq. (19). Now we claim

$$\Pr[\mathrm{G}_6(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{win} \text{ at line 16}] \le (L-1) \cdot Q_c \cdot 2^{-k/5} \,. \tag{20}$$

The collision of line 16 occurs if

$$\begin{aligned} &\mathbf{T}_{r',0}[i][\cdot] + (\mathrm{R}[1, x_{r',i}] + \mathrm{R}[1, x]) \cdot \mathbf{diag}(s_{r',1}) \\ &= \mathbf{T}_{u,0}[v][\cdot] + (\mathrm{R}[1, x_{u,v}] + \mathrm{R}[1, x]) \cdot \mathbf{diag}(s_{u,1}) \,. \end{aligned}$$

We now consider some cases.

Case 1: $u \ne r'$. Line 12 implies $x \ne x_{r',i}$ so due to line 21 we have $\mathbf{w}_{\mathrm{H}}(\mathrm{R}[1, x] + \mathrm{R}[1, x_{r',i}]) \ge k/5$. Then as above, the choice of $s_{r',1}$ from $\{0,1\}^k$, and the fact that the adversary has no information about it, means that the probability of the collision is at most $(L - n_{r'}) \cdot 2^{-k/5}$ due to Lemma 6.5 and because there are at most $L - n_{r'}$ possible values for collision.

Case 2: $u = r'$. Then the collision looks like

$$\mathbf{T}_{r',0}[i][\cdot] + \mathbf{T}_{r',0}[v][\cdot] = (\mathrm{R}[1, x_{r',i}] - \mathrm{R}[1, x_{r',v}]) \cdot \mathbf{diag}(s_{r',1}) \,.$$

We now have subcases.

Case 2.1: $x_{r',i} \ne x_{r',v}$. Then due to a similar analysis as Case 1, the probability of the collision is at most $(n_{r'} - 1) \cdot 2^{-k/5}$.

Case 2.2: $x_{r',i} = x_{r',v}$. Then the collision happens if $\mathbf{T}_{r',0}[i][\cdot] = \mathbf{T}_{r',0}[v][\cdot]$. Line 16 implies $(u, v) \neq (r', i)$ and since $r' = u$ here we have $v \neq i$. Now the choice of $\mathbf{T}_{r',0}$ at line 3 ensures that $\mathbf{T}_{r',0}[i][\cdot] \neq \mathbf{T}_{r',0}[v][\cdot]$, so this case contributes 0.

Accounting for the Q_c queries to CH, the case study above justifies Eq. (20). Putting this together we have

$$\epsilon_6 \leq L \cdot (Q_c + Q_2) \cdot 2^{-k/5} \ .$$

Putting all the above together concludes the proof. ▌

7 JIT-OPRF from VOLE

In this section, we present another unit capacity JIT-OPRF protocol. This one is based on Vector Oblivious Linear Evaluation (VOLE) [11, 57, 12]. It can be used FOR PSI, either directly via our **J2P1** transform, or, for greater server efficiency, in conjunction with a bin-allocation scheme, via our **J2P2** transform. Both transforms are in Section 5.

7.1 Construction

VOLE. Vector Oblivious Linear Evaluation (VOLE) is a vector extension of Oblivious Linear Evaluation (OLE). It can be seen as an arithmetic counterpart of OT. There are several variants of the functionality [11, 57, 12, 19]. We use the basic one. Denoted $\mathsf{F}^{\mathrm{vole}}_{\mathbb{F}}$, it is parameterized by a finite field $\mathbb{F}$, and defined as follows:

$\underline{\mathsf{F}^{\mathrm{vole}}_{\mathbb{F}}(\mathbf{x}, (\mathbf{q}, s))}$ // $\mathbf{x}, \mathbf{q}$ equal-length vectors over $\mathbb{F}$, and $s \in \mathbb{F}$
$\mathbf{t} \leftarrow \mathbf{q} + \mathbf{x} \cdot s$
Return $(\mathbf{t}, \varepsilon)$

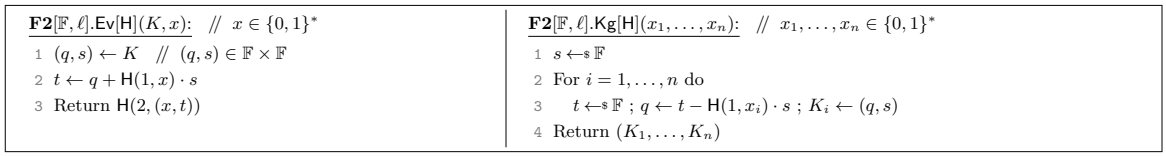
The input of party 1 (client) is vector $\mathbf{x}$ over $\mathbb{F}$. The input of party 2 (server) is a vector $\mathbf{q} \in \mathbb{F}^{|\mathbf{x}|}$ and a scalar $s \in \mathbb{F}$. The functionality outputs a vector $\mathbf{t} \in \mathbb{F}^{|\mathbf{x}|}$ for party 1 and nothing (formally, the empty string) for party 2, where for $i \in [1..n]$ we have $\mathbf{t}[i] = \mathbf{q}[i] + \mathbf{x}[i] \cdot s$.

We will assume given a protocol Π^{v} for the $\mathsf{F}^{\mathrm{vole}}_{\mathbb{F}}$ functionality. We assume it has perfect correctness. We assume it does not use a random oracle. (The extension to the case that it does is easy but one must take care that this RO is independent of any used by the calling JIT-OPRF protocol, which can be ensured with domain separation [4].) Since the $\mathsf{F}^{\mathrm{vole}}_{\mathbb{F}}$ functionality is deterministic, we can use, for Π^{v}, the generic InI security definitions of BRRA [7], recalled in Section 3. Many efficient VOLE protocols exist [11, 57, 13, 60, 59].

CONSTRUCTIONS. We define first a JIT family of functions f and then a corresponding JIT-OPRF Π. Formally, each is built via a transform.

Transform **F2** constructs a JIT family of functions $\mathsf{f} = \mathbf{F2}[\mathbb{F}, \ell]$, where ℓ will be the output length of the function. It will be unit capacity, meaning have capacity $\mathsf{cp} = 1$. It uses RO H where $\mathsf{H}(1, \cdot)\colon \{0,1\}^* \to \mathbb{F}$ and $\mathsf{H}(2, \cdot)\colon (\{0,1\}^* \times \mathbb{F}) \to \{0,1\}^\ell$, so that OS is formally the set of all such functions H. It has $\mathsf{Dom} = \{0,1\}^*$ and $\mathsf{Rng} = \{0,1\}^\ell$. It has $\mathsf{Keys} = \mathbb{F} \times \mathbb{F}$, meaning a key $K = (q, s)$ is a pair of field elements. The evaluation and key-generation algorithms are shown at the top of Figure 14. The output of key-generation is a list of n keys since the capacity is 1.

Transform **J2** constructs a JIT-OPRF protocol $\Pi = \mathbf{J2}[\mathbb{F}, \ell, \Pi^{\mathrm{v}}]$ where $\mathbb{F}, \ell$ are as above and Π^{v} is a protocol for $\mathsf{F}^{\mathrm{vole}}_{\mathbb{F}}$. It has the same random oracles as f. The protocol is shown at the bottom of Figure 14. It uses the technique of [19, 55, 53]. We are adapting this to build a JIT-PRF.



JIT-OPRF Protocol $\Pi = \mathbf{J2}[\mathbb{F},\ell,\Pi^{\mathrm{v}}]$ for realizing functionality $\mathsf{F}_{\mathsf{f}}^{\text{jit-oprf}}$, where $\mathsf{f} = \mathbf{F2}[\mathbb{F},\ell]$

Based on: Protocol Π^{v} for $\mathsf{F}_{\mathbb{F}}^{\mathrm{vole}}$
Party 1 input: $x_1,\ldots,x_n \in \{0,1\}^*$.
Party 2 input: n

1. Party 1 constructs vector $\mathbf{x}$ as $\mathbf{x}[i] \leftarrow \mathsf{H}(1,x_i)$ for $i = 1,\ldots,n$.
2. Party 2 picks $\mathbf{q} \leftarrow\!\!\$\ \mathbb{F}^n$ and $s \leftarrow\!\!\$\ \mathbb{F}$.
3. The parties compute $(\mathbf{t},\varepsilon) \leftarrow \mathsf{F}_{\mathbb{F}}^{\mathrm{vole}}(\mathbf{x},(\mathbf{q},s))$ by running Π^{v}, their inputs being $\mathbf{x}$ and $(\mathbf{q},s)$, respectively.
4. Party 1 outputs $Y_1,\ldots,Y_n$ defined by $Y_i \leftarrow \mathsf{H}(2,(x_i,\mathbf{t}[i]))$ for $i \in [1..n]$.
5. Party 2 outputs $(K_1,\ldots,K_n)$ defined by $K_i \leftarrow (\mathbf{q}[i],s)$ for $i \in [1..n]$.

Figure 14: Top: JIT family of function $\mathsf{f} = \mathbf{F2}[\mathbb{F},\ell]$. Bottom: JIT-OPRF Protocol $\Pi = \mathbf{J2}[\mathbb{F},\ell,\Pi^{\mathrm{v}}]$ for realizing $\mathsf{F}_{\mathsf{f}}^{\text{jit-oprf}}$ based on VOLE. The random oracles here are $\mathsf{H}(1,\cdot)\colon \{0,1\}^* \to \mathbb{F}$ and $\mathsf{H}(2,\cdot)\colon (\{0,1\}^* \times \mathbb{F}) \to \{0,1\}^\ell$.

CORRECTNESS OF **J2**. Consider the protocol at the bottom of Figure 14. Suppose party 1 has input $x_1,\ldots,x_n \in \{0,1\}^*$ and party 2 has input n. Suppose, at step **4**, party 1 outputs $Y_1,\ldots,Y_n$ and, at step **5**, party 2 outputs $(K_1,\ldots,K_n)$. For correctness of the JIT-OPRF $\Pi = \mathbf{J2}[\mathbb{F},\ell,\Pi^{\mathrm{v}}]$, we need to verify that $Y_i = \mathbf{F2}[\mathbb{F},\ell].\mathsf{Ev}[\mathsf{H}](K_i,x_i)$ for all $i \in [1..n]$.

Let $\mathbf{t}$ be as computed at step **3**. Let $i \in [1..n]$. Then we have:

$$\begin{aligned}
\mathbf{t}[i] &= \mathbf{t}[i] - \mathsf{H}(1,x_i) \cdot s + \mathsf{H}(1,x_i) \cdot s \\
&= \mathbf{t}[i] - \mathbf{x}[i] \cdot s + \mathsf{H}(1,x_i) \cdot s && \text{// because } \mathbf{x}[i] = \mathsf{H}(1,x_i) \text{ by step } \mathbf{1} \\
&= \mathbf{q}[i] + \mathsf{H}(1,x_i) \cdot s && \text{// by step } \mathbf{3} \text{ and definition of } \mathsf{F}_{\mathbb{F}}^{\mathrm{vole}}(\mathbf{x},(\mathbf{q},s)) \ .
\end{aligned}$$

Now from step **4** we have $Y_i = \mathsf{H}(2,(x_i,\mathbf{t}[i]))$. From step **5** we know that $K_i = (\mathbf{q}[i],s)$. Now from the top left panel of Figure 14 we can now conclude that $Y_i = \mathbf{F2}[\mathbb{F},\ell].\mathsf{Ev}[\mathsf{H}](K_i,x_i)$ as required.

7.2 Security result

The following implies that $\Pi = \mathbf{J2}[\mathbb{F},\ell,\Pi^{\mathrm{v}}]$ is JIT-OPRF1-secure if Π^{v} is InI-secure for the client (party 1), and JIT-OPRF2-secure if Π^{v} is InI-secure for the server (party 2). After stating the Theorem, we compare the bounds with those of Theorem 6.1. We note that the bounds below correct and update the ones from the preliminary version of this paper [6]. Following that we proceed to the proof, giving this separately for the two parts of the Theorem.

Theorem 7.1 *Let $\mathbb{F}$ be a finite field. Let $\mathsf{f} = \mathbf{F2}[\mathbb{F},\ell]$ be the JIT family of functions shown in Figure 14. Let Π^{v} be a protocol for $\mathsf{F}_{\mathbb{F}}^{\mathrm{vole}}$. Let $\Pi = \mathbf{J2}[\mathbb{F},\ell,\Pi^{\mathrm{v}}]$ be the JIT-OPRF protocol shown in Figure 14. Then:*

1. *Π provides JIT-OPRF1 security if Π^{v} is InI secure for the client: Let $\mathrm{A}_{\mathrm{jit}}$ be an adversary playing game $\mathbf{G}_{\mathsf{f},\Pi}^{\text{jit-oprf-1}}$. Then we can construct an adversary A_{v} playing game $\mathbf{G}_{\mathsf{F}_{\mathbb{F}}^{\mathrm{vole}},\Pi^{\mathrm{v}},1}^{\mathrm{ini}}$ such that*

$$\mathbf{Adv}_{\mathsf{f},\Pi}^{\text{jit-oprf-1}}(\mathrm{A}_{\mathrm{jit}}) \le \mathbf{Adv}_{\mathsf{F}_{\mathbb{F}}^{\mathrm{vole}},\Pi^{\mathrm{v}},1}^{\mathrm{ini}}(\mathrm{A}_{\mathrm{v}}) \ . \tag{21}$$

We will have $\mathrm{Q}^{\mathrm{Run}}(\mathrm{A_v}) = \mathrm{Q}^{\mathrm{CH}}(\mathrm{A_{jit}})$. *The running time of* $\mathrm{A_v}$ *is about the same as that of* $\mathrm{A_{jit}}$.

2. Π *provides JIT-OPRF2 security if* Π^{v} *is InI secure for the server:* *Let* $\mathrm{A_{jit}}$ *be an adversary playing game* $\mathbf{G}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi}$. *Let* $Q_i = \mathrm{Q}^{\mathrm{RO}(i,\cdot)}(\mathrm{A_{jit}})$ *(*$i \in \{1,2\}$*). Let* $Q_T = \mathrm{Q}^{\mathrm{TR}}(\mathrm{A_{jit}})$. *Assume the total number of inputs across all* TR *queries of* $\mathrm{A_{jit}}$ *is* L. *Let* $Q_c = \mathrm{Q}^{\mathrm{CH}}(\mathrm{A_{jit}})$. *Then we can construct an adversary* $\mathrm{A_v}$ *playing game* $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F}^{\mathrm{vole}}_{\mathbb{F}},\Pi^{\mathrm{v}},2}$ *such that*

$$\mathbf{Adv}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi}(\mathrm{A_{jit}}) \le \mathbf{Adv}^{\mathrm{ini}}_{\mathsf{F}^{\mathrm{vole}}_{\mathbb{F}},\Pi^{\mathrm{v}},2} + \delta \;, \tag{22}$$

where

$$\delta = \frac{Q_1^2 + 4L\cdot(Q_2+Q_c) + 2Q_T + L^2}{2\cdot|\mathbb{F}|} \;. \tag{23}$$

We will have $\mathrm{Q}^{\mathrm{Run}}(\mathrm{A_v}) = \mathrm{Q}^{\mathrm{TR}}(\mathrm{A_{jit}})$. *The running time of* $\mathrm{A_v}$ *is about the same as that of* $\mathrm{A_{jit}}$.

NOTE. Eq. (23) is corrected from the proceedings version of this paper [6]. In particular, there is a factor of $(L+1)$ multiplied to the Q_2, Q_c terms which was not present before.

DISCUSSION OF BOUNDS IN THEOREM. The proof of part (1) is tight. In particular, the bound of Eq. (21) does not degrade multiplicatively with the number of CH queries of $\mathrm{A_{jit}}$, as would arise from a hybrid argument. Instead, this is reflected in the number of RUN queries of $\mathrm{A_v}$. This motivates asking how the security of VOLE protocols such as those of [11, 57, 13, 60, 59] degrades with the number of RUN queries, meaning the number of invocations of the protocol. This has not been studied, leading to the interesting open question of whether one can give a tight reduction of the InI security of these VOLE protocols to the underlying hard algebraic problem.

The Eq. (22) bound for part (2) has a computational term, which is the advantage of a VOLE adversary, and a statistical term δ. The bound is tight in the computational term, and all the discussion above applies. It should also be considered tight in the statistical term, unlike Eq. (10). To compare with the latter, let k be the statistical security parameter there, and choose $\mathbb{F} = \mathrm{GF}(2^k)$. Then operations over $\{0,1\}^k$ in the OT-based JIT-OPRF become operations over $\mathrm{GF}(2^k)$ in the VOLE-based JIT-OPRF. Now $|\mathbb{F}| = 2^k$ so the improvement over Eq. (10) is that δ in Eq. (22) is exponentially decreasing in k itself. To see the impact, let us use the same example as in Section 6.2, namely $Q_c = L = 2^{20} \approx 10^6$ and $Q_1 = Q_2 = 2^{60}$ with our target being $\delta \le 2^{-32}$. We again want to pick k as small as possible to allow this. The numerator in Eq. (23) is about 2^{120} so we need $2^{119}\cdot 2^{-k} \le 2^{-32}$ which holds for $k = 151$. This is significantly smaller than the $k = 608$ we derived in Section 6.2. (And also significantly smaller than the $k = 574$ we derived using the generalized result.) It means the VOLE-based JIT-OPRF can be deployed with smaller parameters than the OT-based JIT-OPRF, leading to increased efficiency when the two are compared at the same security level.

7.3 Proof of Theorem 7.1 Part (1)

Proof of Theorem 7.1, Part (1).: Adversary $\mathrm{A_v}$ runs $c' \leftarrow\!\!\$\ \mathrm{A_{jit}}[\mathrm{RO},\mathrm{CH}]$ and returns c' as its own output. Here RO,CH are subroutines that $\mathrm{A_v}$ defines to replace the oracles of the same names that $\mathrm{A_{jit}}$ expects. In defining the first, $\mathrm{A_v}$ uses its RUN oracle along with the RO subroutine that it defines, as follows:

$\mathrm{CH}((x_{1,0},\ldots,x_{n,0}),(x_{1,1},\ldots,x_{n,1}))$:

Games G_0, G_1

$\mathrm{TR}(x_1, \ldots, x_n)$:

$r \leftarrow r + 1$; $n_r \leftarrow n$; $s_{r,0}, s_{r,1} \leftarrow\!\!\$\ \mathbb{F}$

For $i = 1, \ldots, n$ do $\mathbf{x}_r[i] \leftarrow \mathrm{RO}(1, x_i)$

$\mathbf{q}_{r,1} \leftarrow\!\!\$\ \mathbb{F}^n$; $\mathbf{q}_{r,0} \leftarrow \mathbf{q}_{r,1} + \mathbf{x}_r \cdot (s_{r,1} - s_{r,0})$ // $\mathbf{q}_{r,1} + \mathbf{x}_r \cdot s_{r,1} = \mathbf{q}_{r,0} + \mathbf{x}_r \cdot s_{r,0}$

$\mathbf{t}_r \leftarrow \mathbf{q}_{r,1} + \mathbf{x}_r \cdot s_{r,1}$

$\omega_1, \omega_2 \leftarrow\!\!\$\ \Omega$

$(\tau, st_1, st_2) \leftarrow\!\!\$\ \mathbf{Int}_{\Pi^{\mathrm{v}}}(\mathbf{x}_r, (\mathbf{q}_{r,0}, s_{r,0}); \omega_1, \omega_2)$ // Game G_0 only

$(\tau, st_1, st_2) \leftarrow\!\!\$\ \mathbf{Int}_{\Pi^{\mathrm{v}}}(\mathbf{x}_r, (\mathbf{q}_{r,1}, s_{r,1}); \omega_1, \omega_2)$ // Game G_1 only

For $i = 1, \ldots, n$ do $\mathbf{Y}_r[i] \leftarrow \mathrm{RO}(2, (x_i, \mathbf{t}_r[i]))$; $\mathrm{T}[r, i, x_i] \leftarrow \mathbf{Y}_r[i]$

Return $(\mathbf{Y}_r, \tau, \omega_1)$

$\mathrm{CH}(r', i, x)$:

If not $(r' \leq r$ and $i \leq n_{r'})$ then return $\perp$

If $\mathrm{T}[r', i, x] \neq \perp$ then return $\mathrm{T}[r', i, x]$

$t \leftarrow \mathbf{q}_{r',1}[i] + \mathrm{RO}(1, x) \cdot s_{r',1}$; $\mathrm{T}[r', i, x] \leftarrow \mathrm{RO}(2, (x, t))$

Return $\mathrm{T}[r', i, x]$

$\mathrm{RO}(a, X)$:

If $\mathrm{R}[a, X] \neq \perp$ then return $\mathrm{R}[a, X]$

If $a = 1$ then $\mathrm{R}[a, X] \leftarrow\!\!\$\ \mathbb{F}$; Return $\mathrm{R}[a, X]$

$\mathrm{R}[a, X] \leftarrow\!\!\$\ \{0,1\}^\ell$; Return $\mathrm{R}[a, X]$

FINALIZE(c'):

Return $[[c' = 1]]$

Figure 15: Games for the proof of part (2) of Theorem 7.1.

---

1 For $i = 1, \ldots, n$ do $\mathbf{x}_0[i][\cdot] \leftarrow \mathrm{RO}(1, x_{i,0})$; $\mathbf{x}_1[i][\cdot] \leftarrow \mathrm{RO}(1, x_{i,1})$

2 $s \leftarrow\!\!\$\ \mathbb{F}$; $\mathbf{q} \leftarrow\!\!\$\ \mathbb{F}^n$

3 $(\tau, \omega) \leftarrow \textsc{Run}(\mathbf{x}_0, \mathbf{x}_1, (\mathbf{q}, s))$

4 For $i = 1, \ldots, n$ do $K_i \leftarrow (\mathbf{q}[i][\cdot], s)$

5 Return $((K_1, \ldots, K_n), \tau, (\omega, \mathbf{q}, s))$

$\mathrm{A_v}$ defines subroutine RO directly using lazy sampling as follows:

$\mathrm{RO}(a, X)$:

6 If $\mathrm{R}[a, X] \neq \perp$ then return $\mathrm{R}[a, X]$

7 If $a = 1$ then $\mathrm{R}[a, X] \leftarrow\!\!\$\ \mathbb{F}$; Return $\mathrm{R}[a, X]$

8 $\mathrm{R}[a, X] \leftarrow\!\!\$\ \{0,1\}^\ell$; Return $\mathrm{R}[a, X]$

In the CH subroutine, $\mathrm{A_v}$ first constructs n-vectors $\mathbf{x}_0, \mathbf{x}_1$ from the inputs $(x_{1,0}, \ldots, x_{n,0})$ and $(x_{1,1}, \ldots, x_{n,1})$, respectively, using the RO subroutine that it defines. It then picks s and $\mathbf{q}$ as in Π. Using these values, $\mathrm{A_v}$ makes its RUN oracle query in line 3. Note that the output from $\mathsf{F}_{\mathbb{F}}^{\mathrm{vole}}$ for party 2 (adversary here) is ε which makes the functionality output equality check in InI always pass. Finally, $\mathrm{A_v}$ constructs the keys $(K_1, \ldots, K_n)$ using $\mathbf{q}, s$ in line 4 and returns the required output. The challenge bit for $\mathrm{A_v}$ directly determines the challenge bit for $\mathrm{A_{jit}}$. Since $\mathrm{A_v}$ returns the bit output by $\mathrm{A_{jit}}$, we get the result as claimed in Eq. (21).

For resource usage, see that $\mathrm{Q}^{\mathrm{RUN}}(\mathrm{A_v}) = \mathrm{Q}^{\mathrm{CH}}(\mathrm{A_{jit}})$ as $\mathrm{A_v}$ makes a RUN oracle query in each CH query from $\mathrm{A_{jit}}$. Note that $\mathrm{A_v}$ does not have a random oracle. ▮

7.4 Proof of Theorem 7.1 Part (2)

Proof of Theorem 7.1, Part (2).: We use a sequence of games $\mathrm{G}_0, \ldots, \mathrm{G}_7$ that will be executed with $\mathrm{A}_{\mathrm{jit}}$. For brevity it is useful to let

$$\epsilon_i = \Pr[\mathrm{G}_i(\mathrm{A}_{\mathrm{jit}})] \ .$$

Games $\mathrm{G}_0, \mathrm{G}_1$ of Figure 15 start from game $\mathbf{G}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi}$ of Figure 5. They then detail the choices and steps as per the descriptions of f and Π in Figure 14. The games are the same except that line 6 is only in G_0 and line 7 is only in G_1. The difference between these two lines is only in the input for party 2 in the call for Π^{v}, this input being $(\mathbf{q}_{r,0}, s_{r,0})$ for G_0 and $(\mathbf{q}_{r,1}, s_{r,1})$ for G_1. These have been chosen so that the output $\mathbf{q}_{r,0} + \mathbf{x}_r \cdot s_{r,0}$ of party 1 in the first case is the same as its output $\mathbf{t}_r = \mathbf{q}_{r,1} + \mathbf{x}_r \cdot s_{r,1}$ in the second case. Other than at these lines, both games consistently use $(\mathbf{q}_{r,1}, s_{r,1})$ as the server input for the VOLE. The result is that G_1 is equivalent to the $c = 1$ (i.e., "real") case of $\mathbf{G}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi}$. We now write

$$\epsilon_1 = (\epsilon_1 - \epsilon_0) + \epsilon_0 \ .$$

We build adversary A_{v} such that

$$\epsilon_1 - \epsilon_0 \leq \mathbf{Adv}^{\mathrm{ini}}_{\mathsf{F}^{\mathrm{vole}}_{\mathbb{F}},\Pi^{\mathrm{v}},2}(\mathrm{A}_{\mathrm{v}}) \ . \tag{24}$$

Adversary A_{v} runs $c' \leftarrow\!\!\$\ \mathrm{A}_{\mathrm{jit}}[\mathrm{TR}, \mathrm{CH}, \mathrm{RO}]$. It sets $b' \leftarrow c'$, and returns b' as its own output. Here $\mathrm{TR}, \mathrm{CH}, \mathrm{RO}$ are subroutines that A_{v} defines to replace the oracles of the same names that $\mathrm{A}_{\mathrm{jit}}$ expects. In defining the first, A_{v} uses its own RUN oracle, as follows:

$\underline{\mathrm{TR}(x_1, \ldots, x_n)}$:

1 $r \leftarrow r + 1$; $n_r \leftarrow n$; $s_{r,0}, s_{r,1} \leftarrow\!\!\$\ \mathbb{F}$
2 For $i = 1, \ldots, n$ do $\mathbf{x}_r[i] \leftarrow \mathrm{RO}(1, x_i)$
3 $\mathbf{q}_{r,1} \leftarrow\!\!\$\ \mathbb{F}^n$; $\mathbf{q}_{r,0} \leftarrow \mathbf{q}_{r,1} + \mathbf{x}_r \cdot (s_{r,1} - s_{r,0})$ // $\mathbf{q}_{r,1} + \mathbf{x}_r \cdot s_{r,1} = \mathbf{q}_{r,0} + \mathbf{x}_r \cdot s_{r,0}$
4 $\mathbf{t}_r \leftarrow \mathbf{q}_{r,1} + \mathbf{x}_r \cdot s_{r,1}$
5 $(\tau, \omega_1) \leftarrow \mathrm{RUN}((\mathbf{q}_{r,0}, s_{r,0}), (\mathbf{q}_{r,1}, s_{r,1}), \mathbf{x}_r)$
6 For $i = 1, \ldots, n$ do $\mathbf{Y}_r[i] \leftarrow \mathrm{RO}(2, (x_i, \mathbf{t}_r[i]))$; $\mathrm{T}[r, i, x_i] \leftarrow \mathbf{Y}_r[i]$
7 Return $(\mathbf{Y}_r, \tau, \omega_1)$

Above, $\mathbf{q}_{r,1} + \mathbf{x}_r \cdot s_{r,1} = \mathbf{q}_{r,0} + \mathbf{x}_r \cdot s_{r,0}$ ensures that the output of the $\mathsf{F}^{\mathrm{vole}}_{\mathbb{F}}$ for party 1 is the same for the two choices of input given for party 2. Subroutines CH, RO are defined exactly as in games $\mathrm{G}_0, \mathrm{G}_1$, and we do not repeat the code here. (Note that A_{v} does not have a random oracle. Its calls to RO, above and in its CH subroutine, refer to calls to the RO subroutine it itself defines.) Let b be the challenge bit in game $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F}^{\mathrm{vole}}_{\mathbb{F}},\Pi^{\mathrm{v}},2}$. Then, in the execution of $\mathrm{A}_{\mathrm{jit}}$ by A_{v}, the former is seeing the same oracle responses that it would get from G_b. Hence

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{ini}}_{\mathsf{F}^{\mathrm{vole}}_{\mathbb{F}},\Pi^{\mathrm{v}},2}(\mathrm{A}_{\mathrm{v}}) &= \Pr\left[\, b' = 1 \mid b = 1 \,\right] - \Pr\left[\, b' = 1 \mid b = 0 \,\right] \\
&= \Pr\left[\, c' = 1 \mid b = 1 \,\right] - \Pr\left[\, c' = 1 \mid b = 0 \,\right] \\
&= \Pr[\mathrm{G}_1(\mathrm{A}_{\mathrm{jit}})] - \Pr[\mathrm{G}_0(\mathrm{A}_{\mathrm{jit}})] = \epsilon_1 - \epsilon_0 \ ,
\end{aligned}$$

which justifies Eq. (24).

The rest of the proof is devoted to bounding ϵ_0. This is the more difficult part.

Consider game G_2 of Figure 16. Instead of computing $\mathbf{t}_r$ as a function of $\mathbf{q}_{r,1}$ as at line 4 of G_0,

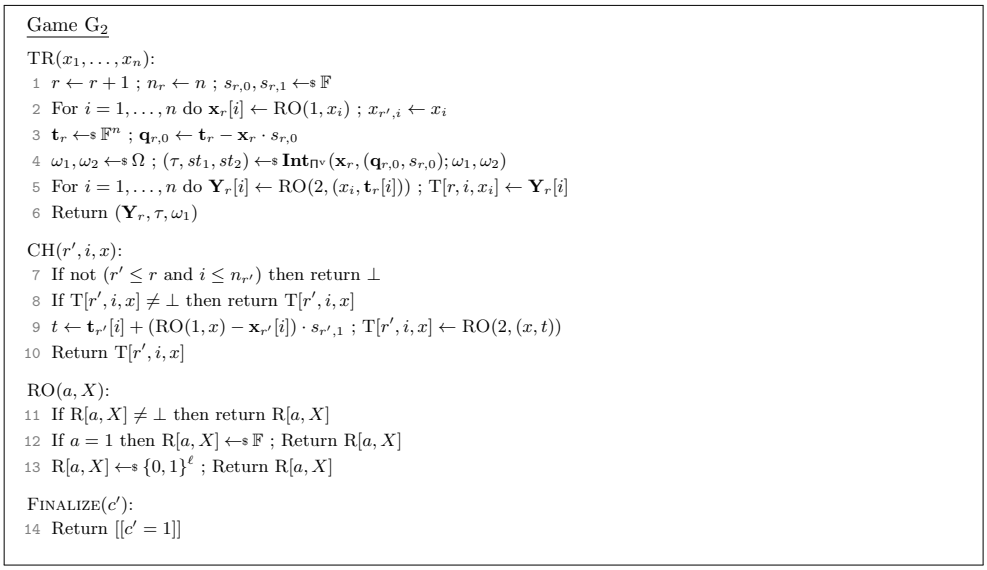


Figure 16: Another game for the proof of part (2) of Theorem 7.1.

game G_2 picks $\mathbf{t}_r$ directly at random at its line 4 and then defines $\mathbf{q}_{r,0}$ as a function of it to ensure that it is the same as at line 3 of G_0. Game G_2 then rewrites the computation of t at line 9 to use $\mathbf{t}_{r'}$ rather than $\mathbf{q}_{r',1}$ as at line 12 of G_0. The games are equivalent, meaning we have

$$\epsilon_0 = \epsilon_2 \ .$$

The benefit from this move is that $s_{1,1}, \ldots, s_{r,1}$ are not used in the execution of Π^{v}. We now need to bound ϵ_2.

Consider games $\mathrm{G}_3, \mathrm{G}_4$ in Figure 17, where G_3 includes the boxed code and G_4 does not. For brevity it is useful to let

$$\begin{aligned}\Delta(u, v, x) &= \mathbf{t}_u[v] + (\mathrm{RO}(1, x) - \mathbf{x}_u[v]) \cdot s_{u,1} \\ &= \mathbf{t}_u[v] + (\mathrm{RO}(1, x) - \mathrm{RO}(1, x_{u,v})) \cdot s_{u,1} \ ,\end{aligned}$$

where $x_{u,v}$ is defined at line 2. The function Δ is computed (explicitly) at lines 9, 12, 18 of the games. Our first claim is that G_3 is equivalent to G_2, meaning

$$\epsilon_2 = \epsilon_3 \ . \tag{25}$$

Let us explain the games and justify Eq. (25). Oracle TR in G_3 is equivalent to its counterpart in G_2. In CH, the reply $\mathrm{T}[r', i, x]$ is, optimistically, chosen at random at line 10. This however could be incorrect, for several reasons. The first, tested at line 11, is that $\mathrm{RO}(2, (x, t))$ has already been queried, in which case, in G_2, it would be the case that $\mathrm{T}[r', i, x]$ equals $\mathrm{R}[2, (x, t)]$. This is ensured in G_3 by the boxed code in line 11. The second, more subtle reason the optimistic choice of line 10 could be incorrect is that the t from line 9 had appeared previously in a CH or TR query, meaning it was equal to $\Delta(u, v, x)$ for some $(u, v) \neq (r', i)$. Indeed, in that case, in G_2, it would be the case that $\mathrm{T}[r', i, x]$ would equal $\mathrm{T}[u, v, x]$. (This is true even if no $\mathrm{RO}(2, \cdot)$ queries have been made so far.) Line 12 tests for such a collision. If it occurs, line 13 sets bad to true, and the boxed code then corrects the optimistic choice of $\mathrm{T}[r', i, x]$. At line 12 it could be that there are multiple pairs (u, v) satisfying the condition, so which do we pick to use in the boxed code at line 13? It does not matter, because the T-values of those choices would be the same. We now discuss RO. In the $a = 1$

Games $\boxed{\mathrm{G}_3}$, G_4

$\mathrm{TR}(x_1, \ldots, x_n)$:
1 $r \leftarrow r+1$; $n_r \leftarrow n$; $s_{r,0}, s_{r,1} \leftarrow\!\!\$\ \mathbb{F}$
2 For $i = 1, \ldots, n$ do $\mathbf{x}_r[i] \leftarrow \mathrm{RO}(1, x_i)$; $x_{r,i} \leftarrow x_i$
3 $\mathbf{t}_r \leftarrow\!\!\$\ \mathbb{F}^n$; $\mathbf{q}_{r,0} \leftarrow \mathbf{t}_r - \mathbf{x}_r \cdot s_{r,0}$
4 $\omega_1, \omega_2 \leftarrow\!\!\$\ \Omega$; $(\tau, st_1, st_2) \leftarrow\!\!\$\ \mathbf{Int}_{\Pi^{\mathrm{v}}}(\mathbf{x}_r, (\mathbf{q}_{r,0}, s_{r,0}); \omega_1, \omega_2)$
5 For $i = 1, \ldots, n$ do $\mathbf{Y}_r[i] \leftarrow \mathrm{RO}(2, (x_i, \mathbf{t}_r[i]))$; $\mathrm{T}[r, i, x_i] \leftarrow \mathbf{Y}_r[i]$
6 Return $(\mathbf{Y}_r, \tau, \omega_1)$

$\mathrm{CH}(r', i, x)$:
7 If not $(r' \le r$ and $i \le n_{r'})$ then return $\perp$
8 If $\mathrm{T}[r', i, x] \neq \perp$ then return $\mathrm{T}[r', i, x]$
9 $t \leftarrow \mathbf{t}_{r'}[i] + (\mathrm{RO}(1, x) - \mathbf{x}_{r'}[i]) \cdot s_{r',1}$
10 $\mathrm{T}[r', i, x] \leftarrow\!\!\$\ \{0,1\}^\ell$
11 If $(\mathrm{R}[2, (x, t)] \neq \perp)$ then $\mathsf{bad} \leftarrow 1$; $\boxed{\mathrm{T}[r', i, x] \leftarrow \mathrm{R}[2, (x, t)]}$
12 If $(\exists\, (u, v)\ :\ (u, v) \neq (r', i)$ and $\mathrm{T}[u, v, x] \neq \perp$ and $t = \mathbf{t}_u[v] + (\mathrm{RO}(1, x) - \mathbf{x}_u[v]) \cdot s_{u,1})$ then
13 $\quad \mathsf{bad} \leftarrow 1$; $\boxed{\mathrm{T}[r', i, x] \leftarrow \mathrm{T}[u, v, x]}$
14 Return $\mathrm{T}[r', i, x]$

$\mathrm{RO}(a, X)$:
15 If $\mathrm{R}[a, X] \neq \perp$ then return $\mathrm{R}[a, X]$
16 If $a = 1$ then $\mathrm{R}[a, X] \leftarrow\!\!\$\ \mathbb{F}$; Return $\mathrm{R}[a, X]$
17 $(x, t) \leftarrow X$; $\mathrm{R}[a, X] \leftarrow\!\!\$\ \{0,1\}^\ell$
18 If $(\exists\, (u, v)\ :\ \mathrm{T}[u, v, x] \neq \perp$ and $t = \mathbf{t}_u[v] + (\mathrm{RO}(1, x) - \mathbf{x}_u[v]) \cdot s_{u,1})$ then
19 $\quad \mathsf{bad} \leftarrow 1$; $\boxed{\mathrm{R}[a, X] \leftarrow \mathrm{T}[u, v, x]}$
20 Return $\mathrm{R}[a, X]$

$\textsc{Finalize}(c')$:
21 Return $[[c' = 1]]$

Figure 17: More games for the proof of part (2) of Theorem 7.1. The boxed code is included only in G_3.

case it is the same as in G_2. In the $a = 2$ case, line 17 parses X as (x, t) and optimistically picks the reply $\mathrm{R}[a, X]$ at random. This is however incorrect if t already occurred as a potential input for $\mathrm{RO}[2, (x, \cdot)]$ in a CH query, meaning if the condition tested at line 18 is true, because in that case, in G_2, the values $\mathrm{R}[a, X]$ and $\mathrm{T}[u, v, x]$ would be the same. Line 19 makes the corresponding correction. Again at line 18 there could be many choices of (u, v) but due to lines 11–13, they would yield the same result for line 19. This concludes our justification of Eq. (25).

Games $\mathrm{G}_3, \mathrm{G}_4$ are identical-until-bad as defined in [9]. By the Fundamental Lemma of Game Playing in [9] (Lemma 2.1), we thus have

$$\epsilon_3 - \epsilon_4 \le \Pr[\mathrm{G}_4(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{bad}] \ . \tag{26}$$

Recall that G_4 does not contain the boxed code. Thus, even though various conditions are tested at lines 11, 12, 18, they do not change the optimistic choices of lines 10, 17, meaning those are what the respective oracles return. The result is that G_4 is equivalent to the $c = 0$ ("random") case of game $\mathbf{G}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi}$. Note this is true even though the transcript τ at line 4 is generated using $s_{r,0}$, meaning we are seeing $(\mathbf{q}_{r,0}, s_{r,0})$ as the server VOLE inputs for this claim. Thus we have

$$\begin{aligned}
&\mathbf{Adv}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi}(\mathrm{A}_{\mathrm{jit}}) = \epsilon_1 - \epsilon_4 \\
&= (\epsilon_1 - \epsilon_0) + (\epsilon_0 - \epsilon_2) + (\epsilon_2 - \epsilon_3) + (\epsilon_3 - \epsilon_4) \\
&\le \mathbf{Adv}^{\mathrm{ini}}_{\mathsf{F}^{\mathrm{vole}}_{\mathbb{F}},\Pi^{\mathrm{v}},1}(\mathrm{A}_{\mathrm{v}}) + \Pr[\mathrm{G}_4(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{bad}] \ .
\end{aligned} \tag{27}$$

To bound $\Pr[\mathrm{G}_4(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{bad}]$, consider games $\mathrm{G}_5, \mathrm{G}_6$ of Figure 18, where G_6 contains the boxed

Games G_5, $\boxed{\mathrm{G}_6}$

$\mathrm{TR}(x_1, \ldots, x_n)$:
1 $r \leftarrow r+1$; $n_r \leftarrow n$; $s_{r,0}, s_{r,1} \leftarrow\!\!\$\ \mathbb{F}$
2 If $(s_{r,1} = 0)$ then $\mathsf{bad} \leftarrow 1$; $\boxed{s_{r,1} \leftarrow\!\!\$\ \mathbb{F} \setminus \{0\}}$
3 For $i = 1, \ldots, n$ do $\mathbf{x}_r[i] \leftarrow \mathrm{RO}(1, x_i)$; $x_{r,i} \leftarrow x_i$
4 $\mathbf{t}_r \leftarrow\!\!\$\ \mathbb{F}^n$
5 For $v = 1, \ldots, n$ do
6 If $(\exists\, u < v\ :\ \mathbf{t}_r[u] = \mathbf{t}_r[v])$ then $\mathsf{bad} \leftarrow 1$; $\boxed{\mathbf{t}_r[v] \leftarrow\!\!\$\ \mathbb{F} \setminus \{\mathbf{t}_r[1], \ldots, \mathbf{t}_r[v-1]\}}$
7 $\mathbf{q}_{r,0} \leftarrow \mathbf{t}_r - \mathbf{x}_r \cdot s_{r,0}$
8 $\omega_1, \omega_2 \leftarrow\!\!\$\ \Omega$; $(\tau, st_1, st_2) \leftarrow\!\!\$\ \mathbf{Int}_{\Pi^{\mathrm{v}}}(\mathbf{x}_r, (\mathbf{q}_{r,0}, s_{r,0}); \omega_1, \omega_2)$
9 For $i = 1, \ldots, n$ do $\mathbf{Y}_r[i] \leftarrow \mathrm{RO}(2, (x_i, \mathbf{t}_r[i]))$; $\mathrm{T}[r, i, x_i] \leftarrow \mathbf{Y}_r[i]$
10 Return $(\mathbf{Y}_r, \tau, \omega_1)$

$\mathrm{CH}(r', i, x)$:
11 If not $(r' \le r$ and $i \le n_{r'})$ then return $\perp$
12 If $\mathrm{T}[r', i, x] \neq \perp$ then return $\mathrm{T}[r', i, x]$
13 $t \leftarrow \mathbf{t}_{r'}[i] + (\mathrm{RO}(1, x) - \mathbf{x}_{r'}[i]) \cdot s_{r',1}$
14 $\mathrm{T}[r', i, x] \leftarrow\!\!\$\ \{0,1\}^\ell$
15 If $(\mathrm{R}[2, (x, t)] \neq \perp)$ then $\mathsf{win} \leftarrow 1$
16 If $(\exists\, (u, v)\ :\ (u, v) \neq (r', i)$ and $\mathrm{T}[u, v, x] \neq \perp$ and $t = \mathbf{t}_u[v] + (\mathrm{RO}(1, x) - \mathbf{x}_u[v]) \cdot s_{u,1})$ then
17 $\mathsf{win} \leftarrow 1$
18 Return $\mathrm{T}[r', i, x]$

$\mathrm{RO}(a, X)$:
19 If $\mathrm{R}[a, X] \neq \perp$ then return $\mathrm{R}[a, X]$
20 If $a = 1$ then
21 $y \leftarrow\!\!\$\ \mathbb{F}$
22 If $(\exists\, X'\ :\ X' \neq X$ and $\mathrm{R}[1, X'] \neq \perp$ and $\mathrm{R}[1, X'] = y)$ then
23 $\mathsf{bad} \leftarrow 1$; $\boxed{y \leftarrow\!\!\$\ \mathbb{F} \setminus \{\, \mathrm{R}[1, X']\ :\ \mathrm{R}[1, X'] \neq \perp \,\}}$
24 $\mathrm{R}[1, X] \leftarrow y$; Return $\mathrm{R}[1, X]$
25 $(x, t) \leftarrow X$; $\mathrm{R}[a, X] \leftarrow\!\!\$\ \{0,1\}^\ell$
26 If $(\exists\, (u, v)\ :\ \mathrm{T}[u, v, x] \neq \perp$ and $t = \mathbf{t}_u[v] + (\mathrm{RO}(1, x) - \mathbf{x}_u[v]) \cdot s_{u,1})$ then $\mathsf{win} \leftarrow 1$
27 Return $\mathrm{R}[a, X]$

$\textsc{Finalize}(c')$:
28 Return win

Figure 18: More games for the proof of part (2) of Theorem 7.1. The boxed code is included only in G_6.

code and G_5 does not. Where G_4 sets bad, games $\mathrm{G}_5, \mathrm{G}_6$ set win, and $\textsc{Finalize}$ returns win. Lines 2,6,22,23 in G_5 can be ignored, since they do not change what the game returns. Thus we have

$$\Pr[\mathrm{G}_4(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{bad}] = \epsilon_5 \ .$$

Games $\mathrm{G}_5, \mathrm{G}_6$ are identical-until-bad as defined in [9]. By the Fundamental Lemma of Game Playing in [9] (Lemma 2.1), we thus have

$$\epsilon_5 = \epsilon_6 + (\epsilon_5 - \epsilon_6) \le \epsilon_6 + \Pr[\mathrm{G}_5(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{bad}] \ . \tag{28}$$

We now have to bound the two terms above. We start with the second, claiming that

$$\Pr[\mathrm{G}_5(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{bad}] \le \frac{Q_T}{|\mathbb{F}|} + \frac{L^2}{2|\mathbb{F}|} + \binom{Q_1}{2} \cdot \frac{1}{|\mathbb{F}|} \ . \tag{29}$$

The three terms above correspond to the setting of bad at lines 2, 6, 23, respectively. We justify Eq. (29) by considering each of these in turn. The probability that line 2 sets bad is $1/|\mathbb{F}|$, so this line contributes at most $Q_T/|\mathbb{F}|$ overall. The probability that line 6 sets bad is $\binom{n}{2}/|\mathbb{F}|$ so overall it

Game G_7

$\mathrm{TR}(x_1, \ldots, x_n)$:
1 $r \leftarrow r+1$; $n_r \leftarrow n$; $s_{r,0} \leftarrow \mathbb{F}$; $s_{r,1} \leftarrow\!\!\$\ \mathbb{F} \setminus \{0\}$
2 For $i = 1, \ldots, n$ do $\mathbf{x}_r[i] \leftarrow \mathrm{RO}(1, x_i)$; $x_{r,i} \leftarrow x_i$
3 For $v = 1, \ldots, n$ do $\mathbf{t}_r[v] \leftarrow\!\!\$\ \mathbb{F} \setminus \{\mathbf{t}_r[1], \ldots, \mathbf{t}_r[v-1]\}$
4 $\mathbf{q}_{r,0} \leftarrow \mathbf{t}_r - \mathbf{x}_r \cdot s_{r,0}$
5 $\omega_1, \omega_2 \leftarrow\!\!\$\ \Omega$; $(\tau, st_1, st_2) \leftarrow\!\!\$\ \mathbf{Int}_{\Pi^{\mathrm{v}}}(\mathbf{x}_r, (\mathbf{q}_{r,0}, s_{r,0}); \omega_1, \omega_2)$
6 For $i = 1, \ldots, n$ do $\mathbf{Y}_r[i] \leftarrow \mathrm{RO}(2, (x_i, \mathbf{t}_r[i]))$; $\mathrm{T}[r, i, x_i] \leftarrow \mathbf{Y}_r[i]$
7 Return $(\mathbf{Y}_r, \tau, \omega_1)$

$\mathrm{CH}(r', i, x)$:
8 If not $(r' \le r$ and $i \le n_{r'})$ then return $\perp$
9 If $\mathrm{T}[r', i, x] \neq \perp$ then return $\mathrm{T}[r', i, x]$
10 $t \leftarrow \mathbf{t}_{r'}[i] + (\mathrm{RO}(1, x) - \mathbf{x}_{r'}[i]) \cdot s_{r',1}$
11 $\mathrm{T}[r', i, x] \leftarrow\!\!\$\ \{0,1\}^{\ell}$
12 If $(\mathrm{R}[2, (x, t)] \neq \perp)$ then $\mathsf{win} \leftarrow 1$
13 If $(\exists\, (u, v) \ :\ (u, v) \neq (r', i)$ and $\mathrm{T}[u, v, x] \neq \perp$ and $t = \mathbf{t}_u[v] + (\mathrm{RO}(1, x) - \mathbf{x}_u[v]) \cdot s_{u,1})$ then
14 $\quad \mathsf{win} \leftarrow 1$
15 Return $\mathrm{T}[r', i, x]$

$\mathrm{RO}(a, X)$:
16 If $\mathrm{R}[a, X] \neq \perp$ then return $\mathrm{R}[a, X]$
17 If $a = 1$ then $\mathrm{R}[1, X] \leftarrow\!\!\$\ \mathbb{F} \setminus \{\, \mathrm{R}[1, X'] \ :\ \mathrm{R}[1, X'] \neq \perp \,\}$; Return $\mathrm{R}[1, X]$
18 $(x, t) \leftarrow X$; $\mathrm{R}[a, X] \leftarrow\!\!\$\ \{0,1\}^{\ell}$
19 If $(\exists\, (u, v) \ :\ \mathrm{T}[u, v, x] \neq \perp$ and $t = \mathbf{t}_u[v] + (\mathrm{RO}(1, x) - \mathbf{x}_u[v]) \cdot s_{u,1})$ then $\mathsf{win} \leftarrow 1$
20 Return $\mathrm{R}[a, X]$

$\textsc{Finalize}(c')$:
21 Return win

Figure 19: Final game for the proof of part (2) of Theorem 7.1.

contributes

$$\frac{1}{|\mathbb{F}|} \cdot \sum_{i=1}^{r} \binom{n_i}{2} = \frac{1}{2|\mathbb{F}|} \cdot \sum_{i=1}^{r} n_i(n_i - 1) \le \frac{1}{2|\mathbb{F}|} \cdot \sum_{i=1}^{r} n_i^2$$
$$\le \frac{1}{2|\mathbb{F}|} \cdot \left(\sum_{i=1}^{r} n_i \right)^2 \le \frac{L^2}{2|\mathbb{F}|} \ ,$$

which justifies the second term of Eq. (29). Line 23 results in bad being set if two distinct queries to $\mathrm{RO}(1, \cdot)$ have the same result, and the probability of this is at most the last term of Eq. (29). This completes the justification of the latter equation.

Now we have to bound ϵ_6. Towards this, game G_7 of Figure 19 rewrites G_6 in simplified form. We have

$$\epsilon_6 = \epsilon_7 \ .$$

We now bound ϵ_7. A key observation towards this is that the adversary has no information about $s_{1,1}, \ldots, s_{r,1}$. We first claim

$$\Pr[\mathrm{G}_7(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{win} \text{ at line } 12] \le \frac{Q_c}{|\mathbb{F}| - 1} \tag{30}$$

$$\Pr[\mathrm{G}_7(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{win} \text{ at line } 19] \le \frac{L \cdot Q_2}{|\mathbb{F}| - 1} \ . \tag{31}$$

Observe that at line 11 we will have $x \neq x_{r',i}$ due to line 9. Due to line 17, it must be that

$\mathrm{R}[1,x] - \mathrm{R}[1,x_{r',i}] \neq 0$. Then the probability that $\mathrm{A}_{\mathrm{jit}}$ makes a query $\mathsf{H}(2,(x, \mathbf{t}_{r'}[i] + (\mathrm{RO}(1,x) - \mathbf{x}_{r'}[i]) \cdot s_{r',1}))$ is at most the probability that it computes $(\mathrm{R}[1,x] - \mathrm{R}[1,x_{r',i}]) \cdot s_{r',1}$. But above we noted that $\mathrm{A}_{\mathrm{jit}}$ has no information about $s_{r',1}$, so this probability is at most $1/(|\mathbb{F}|-1)$. Accounting for the Q_c queries to CH, this justifies Eq. (30). Similarly at line 19 we will have $x \neq x_{u,v}$ since $\mathrm{T}[u,v,x] \neq \perp$ as per the test on the same line, and thus a similar argument holds. However, here, there are at most L values, instead of 1, that would lead to a favorable outcome (picking a correct t value that satisfies the condition) for $\mathrm{A}_{\mathrm{jit}}$. Thus, probability that $\mathrm{A}_{\mathrm{jit}}$ sets the win flag in a single $\mathrm{RO}(2,\cdot)$ query is at most $L/(|\mathbb{F}|-1)$. Accounting for the Q_2 queries to $\mathrm{RO}(2,\cdot)$, this justifies Eq. (31). Now we claim

$$\Pr[\mathrm{G}_7(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{win} \text{ at line 13}] \leq \frac{(L-1) \cdot Q_c}{|\mathbb{F}|-1} \ . \tag{32}$$

The collision of line 13 occurs if

$$\mathbf{t}_{r'}[i] + (\mathrm{R}[1,x] - \mathrm{R}[1,x_{r',i}]) \cdot s_{r',1} = \mathbf{t}_u[v] + (\mathrm{R}[1,x] - \mathrm{R}[1,x_{u,v}]) \cdot s_{u,1} \ .$$

We now consider some cases.

Case 1: $u \neq r'$. Line 9 implies $x \neq x_{r',i}$ so due to line 17 we have $\mathrm{R}[1,x] - \mathrm{R}[1,x_{r',i}] \neq 0$. Then as above, the choice of $s_{r',1}$ from $\mathbb{F} \setminus \{0\}$, and the fact that the adversary has no information about it, means that the probability of the collision is at most $(L - n_{r'})/(|\mathbb{F}|-1)$. Here, there are $(L - n_{r'})$ favorable values as $u \neq r'$.

Case 2: $u = r'$. Then the collision looks like

$$\mathbf{t}_{r'}[i] - \mathbf{t}_{r'}[v] = (\mathrm{R}[1,x_{r',i}] - \mathrm{R}[1,x_{r',v}]) \cdot s_{r',1} \ .$$

We now have subcases.

Case 2.1: $x_{r',i} \neq x_{r',v}$. Then with similar analysis in Case 1 and $n_{r'}$ favorable values, the probability of the collision is at most $(n_{r'}-1)/(|\mathbb{F}|-1)$.

Case 2.2: $x_{r',i} = x_{r',v}$. Then the collision happens if $\mathbf{t}_{r'}[i] = \mathbf{t}_{r'}[v]$. Line 13 implies $(u,v) \neq (r',i)$ and since $r' = u$ here we have $v \neq i$. Now the choice of $\mathbf{t}_{r'}$ at line 3 ensures that $\mathbf{t}_{r'}[i] \neq \mathbf{t}_{r'}[v]$, so this case contributes 0.

Accounting for the Q_c queries to CH, the case study above justifies Eq. (32). Putting this together and using that $|\mathbb{F}| - 1 \geq |\mathbb{F}|/2$ we have

$$\epsilon_7 \leq \frac{L \cdot (Q_c + Q_2)}{|\mathbb{F}|-1} \leq \frac{2L \cdot (Q_c + Q_2)}{|\mathbb{F}|} \ .$$

Putting all the above together concludes the proof. ∎

8 JIT-OPRF from KVS and VOLE

We present a JIT-PRF, and corresponding JIT-OPRF, built by combining a Linear Key Value Store (LKVS) and VOLE. What distinguishes this from the purely VOLE based constructions of Section 7 is that, while those had unit capacity, the one we give now has infinite (unbounded) capacity. This means it can be directly employed to get fast PSI via our first and simpler **J2P1** transform of Section 5, the resulting PSI protocol having server cost proportional only to the size s_2 of the server set. (Recall that for unit-capacity JIT-OPRFs, **J2P1** yields a PSI protocol with server cost $s_1 \cdot s_2$ where s_1 is the size of the client set. One can use a bin-allocation scheme with

J2P2 to reduce this to $c \cdot s_2$, but, while c is a constant, it will not be 1.) The construction in this section is based on the fast PSI protocols of [19, 55, 53, 10]. We start by recalling the definition of a Linear Key Value Store (LKVS) [19].

8.1 Construction

Linear key value stores. Garimella et al. [19] introduced Linear Key Value Stores (LKVSs) as a generalization of the input-encoding procedures that are used in PSI protocols. They gave several constructions. We formalize LKVSs in a somewhat different way, and with a stronger correctness condition, than in prior work. A linear key value store (LKVS) KS specifies parameters and algorithms as follows:

- $\mathbb{F}$— A finite field. For equal-length vectors $\mathbf{v}_1, \mathbf{v}_2$ over $\mathbb{F}$, we let $\langle \mathbf{v}_1, \mathbf{v}_2 \rangle$ denote their inner product over $\mathbb{F}$.
- OS— An oracle space from which the random oracle G used by other algorithms is drawn.
- $kp \leftarrow\!\!\$\ \mathsf{SU}[\mathsf{G}]$— The setup algorithm returns setup parameters $kp \in \{0,1\}^*$. We say that the scheme has no setup if it is always the case that $kp = \varepsilon$.
- $\boldsymbol{\eta} \leftarrow \mathsf{e}[\mathsf{G}](kp, x, n)$— The deterministic embedding algorithm takes as input setup parameters, a string $x \in \{0,1\}^*$ (called a "key" in this context) and an integer n. The output $\boldsymbol{\eta} \in \mathbb{F}^{\mathsf{el}(n)}$ is a length $\mathsf{el}(n)$ vector over $\mathbb{F}$ called the embedding of x. The function $\mathsf{el}\colon \mathbb{N} \to \mathbb{N}$ is called the embedding length.
- $(\mathbf{x}, d) \leftarrow\!\!\$\ \mathsf{Enc}[\mathsf{G}](kp, (x_1, v_1), \ldots, (x_n, v_n))$— The encoding algorithm takes as input setup parameters and a list of pairs where $x_1, \ldots, x_n \in \{0,1\}^*$ are distinct strings (keys), $v_1, \ldots, v_n \in \mathbb{F}$ and $n \geq 1$ is arbitrary. It returns a vector $\mathbf{x} \in \mathbb{F}^{\mathsf{el}(n)}$ and a bit d, with $d = 1$ indicating success and $d = 0$ indicating failure (error). It is required that, if $d = 1$, then for all $i \in [1..n]$ we have $\langle \mathsf{e}[\mathsf{G}](kp, x_i, n), \mathbf{x} \rangle = v_i$
- $v \leftarrow \mathsf{Dec}[\mathsf{G}](kp, \mathbf{x}, x, n)$— The deterministic decoding algorithm takes input setup parameters, an encoding $\mathbf{x} \in \mathbb{F}^{\mathsf{el}(n)}$, a key $x \in \{0,1\}^*$ and integer n. It returns $v \in \mathbb{F}$ defined as $v \leftarrow \langle \mathsf{e}[\mathsf{G}](kp, x, n), \mathbf{x} \rangle$.

We observe that if $(\mathbf{x}, d) \leftarrow\!\!\$\ \mathsf{Enc}[\mathsf{G}](kp, (x_1, v_1), \ldots, (x_n, v_n))$ and $d = 1$ then, for all $i \in [n]$, we have $\mathsf{Dec}[\mathsf{G}](kp, \mathbf{x}, x, n) = v_i$.

We define correctness error of a LKVS scheme KS using the game $\mathbf{G}^{\text{corr-kvs}}_{\mathsf{KS}}$ shown in Figure 20. The advantage for an adversary A is defined as $\mathbf{Adv}^{\text{corr-kvs}}_{\mathsf{KS}}(\mathrm{A}) = \Pr[\mathbf{G}^{\text{corr-kvs}}_{\mathsf{KS}}]$. Adversary A has access to an Enc oracle. It queries it with inputs $(x_1, v_1), \ldots, (x_n, v_n)$ and wins if encoding fails. The adversary also has access to the random oracle G that is picked by the game. We say KS is perfectly correct if $\mathbf{Adv}^{\text{corr-kvs}}_{\mathsf{KS}}(\mathrm{A}) = 0$ for all adversaries A.

Encoding Unforgeability (EnUF). Prior works define and use mainly three security properties for LKVS schemes, namely *obliviousness* [19], *overfitting* [19] and *random decodings* [62]. None of these would be required for the security of the JIT-OPRF that we construct below based on the PSI constructions of [19, 55, 53, 10]. This ultimately means that these properties are not needed to show security of the resulting PSI protocol. Instead, a different, new property that we give here is needed for our results. We call it Encoding Unforgeability (EnUF). Specifically we use EnUF security of the LKVS to show server security of the JIT-OPRF construction below.

Let $\mathcal{V}$ be a public random function. In brief, EnUF requires that it should be hard to find keys $x_1, \ldots, x_n$ such that encoding $(x_1, \mathcal{V}((kp, x_1))), \ldots, (x_n, \mathcal{V}((kp, x_n)))$ results in an output that decodes to $\mathcal{V}((kp, x))$ for some known key $x \notin \{x_1, \ldots, x_n\}$. Here, kp is the setup parameter used for encoding and decoding. Formally, for a LKVS scheme KS, this is defined as a game, $\mathbf{G}^{\text{enuf}}_{\mathsf{KS}}$,

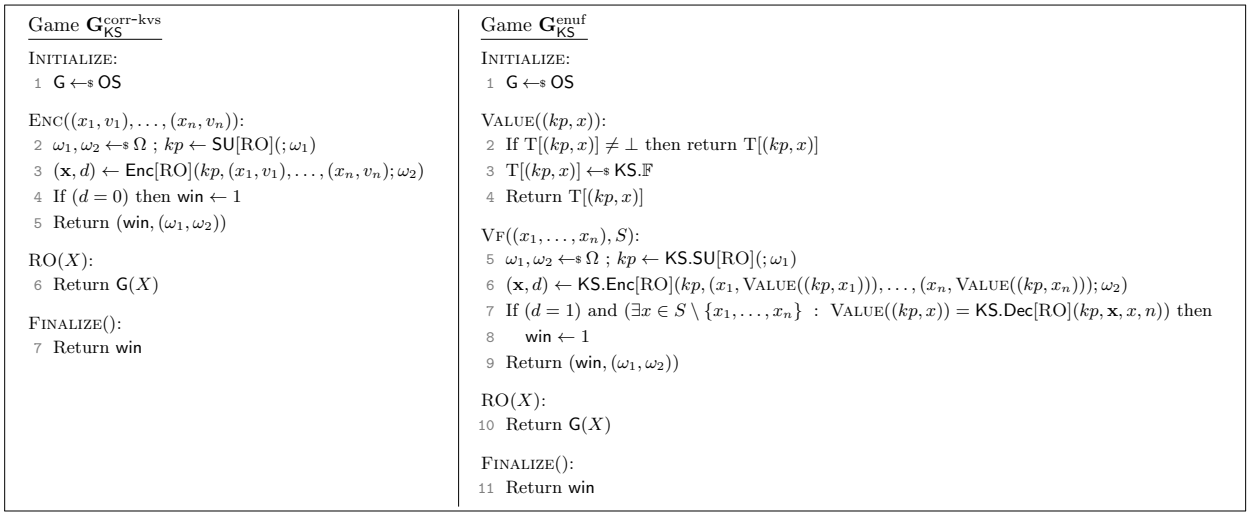


Figure 20: Left: Game for defining correctness error of LKVS KS. Right: Game for defining encoding-unforgeability of LKVS KS.

in Figure 20. An adversary playing the game has access to Value oracle, which models $\mathcal{V}$ as an oracle, and Vf oracle which checks the condition discussed above. The advantage for an adversary A is given as $\mathbf{Adv}^{\text{enuf}}_{\mathsf{KS}}(\mathrm{A}) = \Pr[\mathbf{G}^{\text{enuf}}_{\mathsf{KS}}]$.

We believe that the existing constructions from [19, 55, 53, 10] satisfy this property. However, instead of individually analyzing them, in section 8.5, we give a general transform that converts any LKVS scheme to an LKVS scheme which is EnUF secure (information-theoretically) with minimal overhead.

LKVS Constructions. In the basic approach to LKVS constructions [49, 19, 53, 10], the encoding algorithm, on input $kp, (x_1, v_1), \ldots, (x_n, v_n)$, lets $\mathbf{M}$ be the $n \times \mathsf{el}(n)$ matrix over $\mathbb{F}$ whose i-th row is $\mathsf{e}[\mathsf{G}](kp, x_i, n)$, and defines vector $\mathbf{v} \in \mathbb{F}^n$ by $\mathbf{v}[i] = v_i$ for $i \in [1..n]$. If there is a solution $\mathbf{x} \in \mathbb{F}^{\mathsf{el}(n)}$ to the matrix equation $\mathbf{Mx} = \mathbf{v}$, then it sets $d = 1$ and outputs $(\mathbf{x}, d)$. Else (meaning, the matrix equation does not have a solution) it sets $d = 0$ and outputs $(\mathbf{x}, d)$ for some arbitrary $\mathbf{x}$. So the correctness error arises when a solution to the matrix equation does not exist.

The differences and improvements in the works mentioned above are in the definition of the embedding function e and the methods used to solve the matrix equation. The objective is to have efficient e that allows efficiently solving the resulting matrix equation, while keeping the ratio $\gamma = \mathsf{el}(n)/n$ as low as possible and also having a low (e.g. 2^{-40}) correctness error. This ratio is often called the rate of LKVS scheme but sometimes the reciprocal $1/\gamma$ of γ is defined as the rate [10], in which case, one aims to maximize the rate.

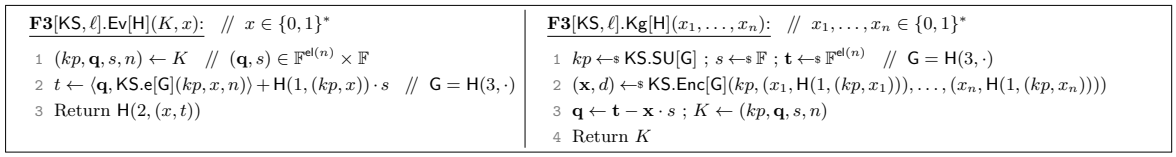
The first construction was given in [49]. In this, the embedding algorithm, on input a key x, outputs an $\mathsf{el}(n)$ length string of Hamming weight $w = 2$. The resulting matrix $\mathbf{M}$ can be seen as a graph consisting of $\mathsf{el}(n)$ vertices and n edges where there is an edge between nodes with column indices having value 1 in each row. In [49], the output of embedding function is constructed using two hash functions which on input x output the two indices, respectively, where the string value would be 1 (with other places having value 0). For correctness and security results, the two hash functions are seen as random functions having distinct outputs in $[1..\mathsf{el}(n)]$ for the same input x. The solution vector $\mathbf{x}$ for the resulting matrix equation is computed using a *peeling* process. They achieve $\gamma \approx 2.4$ for the desired 2^{-40} correctness error and $n \approx 2^{20}$. This approach was improved in [19] who used a similar embedding function and equation solving method but with the output

string of embedding function having $w = 3$, so, $\mathbf{M}$ can now be seen as a hyper-graph. Additionally, they apply a correctness amplification method to the encoding algorithm, leading to $\gamma \approx 1.32$ for similar desired properties and input sizes. The construction in [53] improves the peeling process in previous constructions through their matrix triangulation method and achieves $\gamma \approx 1.23$ while having a faster overall encoding algorithm since correctness amplification is not needed. Another improvement was made in the construction in [10]. Here, the embedding function applied on a key constructs a sparse vector that only has a short band (consecutive elements) of random bits with all other elements being 0. This achieves the best ratio yet of $\gamma \approx 1.05$ albeit with slower encoding and decoding algorithms than [53] since the same matrix equation solving method cannot be applied. All the above constructions have correctness error. A natural, perfectly correct LKVS is described in [19] as well. The idea is to use polynomial interpolation on the inputs to construct a polynomial P such that $P(x_i) = v_i$ for all $i \in [1..n]$. In terms of the matrix equation above, here the solution vector $\mathbf{x}$ is the coefficients of P and the embedding function on key x_i outputs $(1, x_i, x_i^2, \ldots, x_i^{n-1})$. This polynomial can always be computed as long as $|\mathbb{F}| > n$. Note that here $\gamma = 1$ but the encoding (polynomial interpolation) and decoding (polynomial evaluation) algorithms are not as efficient as the above constructions.

The LKVS schemes in literature do not use setup parameters kp. The random functions used in constructions are parameters made available to the encoding and decoding algorithms. We define the setup algorithm SU as a way of explicitly defining how these random functions are picked. One natural way, that is used in literature, is to model them as random oracles directly. However, similar to the subtlety discussed for defining random hash functions for cuckoo hashing (in section 5.2), here too, this would lead to issues as we explain below. An implicit assumption in correctness results and security proofs involving these LKVS constructions (for e.g. in PSI) is that the inputs to encoding algorithm are independent of these random functions. However, if the random functions are defined as in prior work, this may not be case in our definition of correctness which explicitly provides the random oracle to the adversary and allows adaptive queries to the Enc oracle. For example, an adversary playing the correctness game may query the random oracle any number of times to figure out favorable keys that would lead to an encoding failure resulting in high advantage for the adversary. As in section 5.2, we suggest a salting based work around for this problem. The idea is that any random function h_i (usually two are needed) required by a construction is defined using the random oracle G as $h_i = \mathsf{G}(\sigma_i, \cdot)$ where σ_i is a sufficiently large (say 256 bits) uniformly picked string. These salts would be computed in the setup algorithm $\mathsf{SU}[\mathsf{G}]$ and returned in kp. Thus, they will be freshly picked in each Enc oracle query of $\mathbf{G}^{\text{corr-kvs}}_{\mathsf{KS}}$. Since it is unlikely that adversary makes a random oracle query with the correct salt, the keys it uses for Enc oracle query would be independent of the random functions picked, as required. Due to the above, our definition explicitly contains a setup algorithm whose output is made available to the other algorithms. We will use a similar salting technique in the setup parameters in section 8.5 to construct an EnUF secure LKVS scheme from any LKVS scheme.

Construction. We define first a JIT family of functions f and then a corresponding JIT-OPRF $\mathsf{\Pi}$. Formally, each is built via a transform.

Transform **F3** constructs a JIT family of functions $\mathsf{f} = \mathbf{F3}[\mathsf{KS}, \ell]$, where KS is a LKVS scheme as above, and ℓ will be the output length of the function. It will be unbounded capacity, meaning have capacity $\mathsf{cp} = \infty$. We denote by $\mathbb{F}$ the field associated to KS. The JIT family f uses RO H where $\mathsf{H}(1, \cdot)\colon (\{0,1\}^* \times \{0,1\}^*) \to \mathbb{F}$ and $\mathsf{H}(2, \cdot)\colon (\{0,1\}^* \times \mathbb{F}) \to \{0,1\}^\ell$ and in addition $\mathsf{H}(3, \cdot) \in \mathsf{KS.OS}$ is a random oracle for KS, so that $\mathsf{f.OS}$ is formally the set of all such functions H. It has $\mathsf{Dom} = \{0,1\}^*$ and $\mathsf{Rng} = \{0,1\}^\ell$. A key has the form $K = (kp, \mathbf{q}, s, n)$ where $\mathbf{q} \in \mathbb{F}^{\mathsf{el}(n)}$ and $s \in \mathbb{F}$ and $n \geq 1$. The evaluation and key-generation algorithms are shown at the top of Figure 21. In the



JIT-OPRF Protocol $\Pi = \mathbf{J3}[\mathsf{KS},\ell,\Pi^{\mathrm{v}}]$ for realizing functionality $\mathsf{F}_{\mathsf{f}}^{\text{jit-oprf}}$, where $\mathsf{f} = \mathbf{F3}[\mathsf{KS},\ell]$

Based on: Linear Key Value Store KS and Protocol Π^{v} for $\mathsf{F}_{\mathbb{F}}^{\mathrm{vole}}$
Party 1 input: Distinct $x_1,\ldots,x_n \in \{0,1\}^*$.
Party 2 input: n

1. Party 1 lets $kp \leftarrow\!\!\$\ \mathsf{SU}$ and $(\mathbf{x},d) \leftarrow\!\!\$\ \mathsf{Enc}[\mathsf{G}](kp,(x_1,\mathsf{H}(1,(kp,x_1))),\ldots,(x_n,\mathsf{H}(1,(kp,x_n))))$. If $d=1$ it sends kp to party 2. Else it sends d to party 2 (as an indication to abort), rejects (formally, sets $st_1.\mathrm{dec} = 0$ and $st_1.\mathrm{out} = \perp$, where st_1 is its state) and terminates.
2. Party 2 picks $\mathbf{q} \leftarrow\!\!\$\ \mathbb{F}^{\mathsf{el}(n)}$ and $s \leftarrow\!\!\$\ \mathbb{F}$.
3. The parties compute $(\mathbf{t},\varepsilon) \leftarrow \mathsf{F}_{\mathbb{F}}^{\mathrm{vole}}(\mathbf{x},(\mathbf{q},s))$ by running Π^{v}, their inputs being $\mathbf{x}$ and $(\mathbf{q},s)$, respectively.
4. Party 1 defines $y_i \leftarrow \langle \mathbf{t}, \mathsf{e}[\mathsf{G}](kp,x_i,n)\rangle$ and $Y_i \leftarrow \mathsf{H}(2,(x_i,y_i))$ for $i \in [1..n]$, and outputs $Y_1,\ldots,Y_n$.
5. Party 2 outputs $K \leftarrow (kp,\mathbf{q},s,n)$.

Figure 21: Top: JIT family of function $\mathsf{f} = \mathbf{F3}[\mathsf{KS},\ell]$. Bottom: JIT-OPRF Protocol $\Pi = \mathbf{J3}[\mathsf{KS},\ell,\Pi^{\mathrm{v}}]$ for realizing $\mathsf{F}_{\mathsf{f}}^{\text{jit-oprf}}$ based on LKVS and VOLE. The random oracles here are $\mathsf{H}(1,\cdot)\colon (\{0,1\}^* \times \{0,1\}^*) \to \mathbb{F}$ and $\mathsf{H}(2,\cdot)\colon (\{0,1\}^* \times \mathbb{F}) \to \{0,1\}^\ell$ and $\mathsf{G} = \mathsf{H}(3,\cdot) \in \mathsf{KS.OS}$.

code, we let $\mathsf{G} = \mathsf{H}(3,\cdot)$.

Transform **J3** constructs a JIT-OPRF protocol $\Pi = \mathbf{J3}[\mathsf{KS},\ell,\Pi^{\mathrm{v}}]$ where KS,ℓ are as above and Π^{v} is a protocol for $\mathsf{F}_{\mathbb{F}}^{\mathrm{vole}}$ where $\mathbb{F}$ is the field underlying KS. The protocol is shown in the bottom part of Figure 21. It has the same random oracles as f. Here we have adapted the PSI protocols of [19, 53, 10] to build instead a JIT-PRF and the corresponding JIT-OPRF.

Note that $\Pi = \mathbf{J3}[\mathsf{KS},\ell,\Pi^{\mathrm{v}}]$ requires (Figure 21) that the inputs $x_1,\ldots,x_n$ of party 1 are distinct. This is needed by the LKVS. This condition will be met when the JIT-OPRF is used for PSI as in Section 5 since $x_1,\ldots,x_n$ will be the elements of the client set. The condition is in any case wlog.

CORRECTNESS OF **J3**. Consider the protocol at the bottom of Figure 21. Suppose party 1 has input $x_1,\ldots,x_n \in \{0,1\}^*$ and party 2 has input n. Suppose, at step 4, party 1 outputs $Y_1,\ldots,Y_n$ and, at step 5, party 2 outputs $K = (kp,\mathbf{q},s,n)$. For simplicity assume that KS is perfectly correct. Now for correctness of the JIT-OPRF $\Pi = \mathbf{J3}[\mathsf{KS},\ell,\Pi^{\mathrm{v}}]$, we need to verify that $Y_i = \mathbf{F3}[\mathsf{KS},\ell].\mathsf{Ev}[\mathsf{H}](K,x_i)$ for all $i \in [1..n]$. At step 4 we have

$$\begin{aligned} y_i &= \langle \mathbf{t}, \mathsf{e}[\mathsf{G}](kp,x_i,n)\rangle \\ &= \langle \mathbf{q} + \mathbf{x}\cdot s, \mathsf{e}[\mathsf{G}](kp,x_i,n)\rangle && \text{(by VOLE)} \\ &= \langle \mathbf{q}, \mathsf{e}[\mathsf{G}](kp,x_i,n)\rangle + \langle \mathbf{x}, \mathsf{e}[\mathsf{G}](kp,x_i,n)\rangle \cdot s && \text{(by linearity of inner product)} \\ &= \langle \mathbf{q}, \mathsf{e}[\mathsf{G}](kp,x_i,n)\rangle + \mathsf{H}(1,(kp,x_i)) \cdot s\ , \end{aligned}$$

where the last equality is from the definition of Enc and the fact that $\mathbf{x} = \mathsf{Enc}[\mathsf{G}](kp,(x_1,\mathsf{H}(1,(kp,x_1))),\ldots,(x_n,\mathsf{H}(1,(kp,x_n))))$ from step 1. Now, $Y_i = \mathsf{H}(2,(x_i,y_i))$, and if we then consider the description of $\mathbf{F3}[\mathsf{KS},\ell].\mathsf{Ev}$ in the top-left panel of Figure 21, we see that $Y_i = \mathbf{F3}[\mathsf{KS},\ell].\mathsf{Ev}[\mathsf{G}](K,x_i)$, where $K = (kp,\mathbf{q},s,n)$ is the key output by party 2 in step 4 of the protocol. This, as mentioned above, establishes correctness.

In the above we have assumed for simplicity that KS is perfectly correct. This is not always true and, as discussed above, some schemes have a small correctness error such as 2^{-40}. Having such error will lead to correctness error in the above JIT-OPRF protocol and percolate thence into

the PSI protocols of Section 5.

8.2 Security Result

The following implies that $\Pi = \mathbf{J3}[\mathsf{KS}, \ell, \Pi^{\mathrm{v}}]$ is JIT-OPRF1-secure if Π^{v} is InI-secure for the client (party 1), and JIT-OPRF2-secure if Π^{v} is InI-secure for the server (party 2). We note that for this result, no security property or assumption is needed for the LKVS scheme KS. In the Theorem, we assume for simplicity that KS is perfectly correct. After the theorem statement, we discuss how correctness error would affect the result.

Theorem 8.1 *Let $\mathbb{F}$ be a finite field. Let KS be a perfectly correct LKVS scheme with associated field $\mathbb{F}$. Let $\mathsf{f} = \mathbf{F3}[\mathsf{KS}, \ell]$ be the JIT family of functions shown in Figure 21. Let Π^{v} be a protocol for $\mathsf{F}_{\mathbb{F}}^{\mathrm{vole}}$. Let $\Pi = \mathbf{J3}[\mathsf{KS}, \ell, \Pi^{\mathrm{v}}]$ be the JIT-OPRF protocol shown in Figure 21. Then:*

1. *Π provides JIT-OPRF1 security if Π^{v} is InI secure for the client: Let $\mathrm{A}_{\mathrm{jit}}$ be an adversary playing game $\mathbf{G}^{\text{jit-oprf-1}}_{\mathsf{f},\Pi}$. Then we can construct an adversary A_{v} playing game $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F}_{\mathbb{F}}^{\mathrm{vole}},\Pi^{\mathrm{v}},1}$ such that*

$$\mathbf{Adv}^{\text{jit-oprf-1}}_{\mathsf{f},\Pi}(\mathrm{A}_{\mathrm{jit}}) \leq \mathbf{Adv}^{\mathrm{ini}}_{\mathsf{F}_{\mathbb{F}}^{\mathrm{vole}},\Pi^{\mathrm{v}},1}(\mathrm{A}_{\mathrm{v}}) \ . \tag{33}$$

   We will have $\mathrm{Q}^{\mathrm{RUN}}(\mathrm{A}_{\mathrm{v}}) = \mathrm{Q}^{\mathrm{CH}}(\mathrm{A}_{\mathrm{jit}})$. The running time of A_{v} is about the same as that of $\mathrm{A}_{\mathrm{jit}}$.

2. *Π provides JIT-OPRF2 security if Π^{v} is InI secure for the server and KS is EnUF secure: Let $\mathrm{A}_{\mathrm{jit}}$ be an adversary playing game $\mathbf{G}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi}$. Let $Q_i = \mathrm{Q}^{\mathrm{RO}(i,\cdot)}(\mathrm{A}_{\mathrm{jit}})$ $(i \in \{1,2,3\})$. Let $Q_T = \mathrm{Q}^{\mathrm{TR}}(\mathrm{A}_{\mathrm{jit}})$. Let $Q_c = \mathrm{Q}^{\mathrm{CH}}(\mathrm{A}_{\mathrm{jit}})$. Then we can construct adversaries A_{v} and A_{en} playing games $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F}_{\mathbb{F}}^{\mathrm{vole}},\Pi^{\mathrm{v}},2}$ and $\mathbf{G}^{\mathrm{enuf}}_{\mathsf{KS}}$, respectively, such that*

$$\mathbf{Adv}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi}(\mathrm{A}_{\mathrm{jit}}) \leq \mathbf{Adv}^{\mathrm{ini}}_{\mathsf{F}_{\mathbb{F}}^{\mathrm{vole}},\Pi^{\mathrm{v}},2}(\mathrm{A}_{\mathrm{v}}) + \mathbf{Adv}^{\mathrm{enuf}}_{\mathsf{KS}}(\mathrm{A}_{\mathrm{en}}) + \delta \ , \tag{34}$$

   where

$$\delta = \frac{Q_T \cdot (1 + Q_1 + 2Q_2 + 2Q_c)}{|\mathbb{F}|} \ . \tag{35}$$

   We will have $\mathrm{Q}^{\mathrm{RUN}}(\mathrm{A}_{\mathrm{v}}) = \mathrm{Q}^{\mathrm{VF}}(\mathrm{A}_{\mathrm{en}}) = \mathrm{Q}^{\mathrm{TR}}(\mathrm{A}_{\mathrm{jit}})$ and $\mathrm{Q}^{\mathrm{VALUE}}(\mathrm{A}_{\mathrm{en}}) = Q_1$. The running times of A_{v} and A_{en} are about the same as that of $\mathrm{A}_{\mathrm{jit}}$.

DISCUSSION. The above assumes for simplicity that KS is perfectly correct. If it isn't, an additional $2 \cdot \mathbf{Adv}^{\text{corr-kvs}}_{\mathsf{KS}}(\mathrm{A}_{\mathrm{corr}})$ term would be added in Eq. (33), for an adversary $\mathrm{A}_{\mathrm{corr}}$ that the proof would construct. Eq. (34) would not change.

Similar to Eq. (22), the bound in Eq. (34) also has computational and statistical terms. Here there is an additional computational term for the EnUF security of the underlying LKVS scheme KS. The computational bound is tight for InI security of Π^{v} as before. It is also tight for the EnUF security of KS. Later in section 8.5, we will give a universal transform which converts any KS into one with information-theoretic EnUF security. Thus, we would be able to see this computational term as statistical as well. Compared to Eq. (22), the statistical term, δ, in Eq. (34) is smaller when using the same values for Q_c, Q_T, Q_1, Q_2. In particular, there is no Q_1^2 term as in Eq. (23) and no dependency on L (total number of inputs across all TR queries). This allows implementations to use smaller parameters, for example smaller sizes of finite fields $\mathbb{F}$. To see this concretely, we can use the same example as before (in sections 7.2, 6.2), where $Q_T = 2^{10}, Q_c = 2^{20}$ and $Q_1 = Q_2 = 2^{60}$

(we used $L = 2^{20}$ in previous sections, so here assuming $Q_T = 2^{10}$ is reasonable). We want to figure out the smallest value of statistical security parameter k, which affects size of finite field as $|\mathbb{F}| = 2^k$, that makes $\delta \leq 2^{-32}$. Our example values above make the numerator in Eq. (35) around 2^{71}. We get $2^{71} \cdot 2^{-k} \leq 2^{-32}$ which makes $k \geq 103$. This is much lower than the values we got in sections 7.2, 6.2 which were 151 and 574 (using the generalized result), respectively. With such small lower bound for k, the size of $\mathbb{F}$ is likely to be dictated by the computational term $\mathbf{Adv}^{\text{ini}}_{\mathsf{F}^{\text{vole}}_{\mathbb{F}},\Pi^{\text{v}},2}(\mathrm{A}_{\text{v}})$. Thus, we can think of the statistical bound to be tight (insignificant) as well.

8.3 Proof for Theorem 8.1 Part (1)

Proof of Theorem 8.1, Part (1): Adversary A_{v} runs $c' \leftarrow\!\$\ \mathrm{A}_{\text{jit}}[\mathrm{RO}, \mathrm{CH}]$ and returns c' as its own output. Here RO, CH are subroutines that A_{v} defines to replace the oracles of the same names that A_{jit} expects. In defining the first, A_{v} uses its RUN oracle along with the RO subroutine that it defines, as follows:

$\underline{\mathrm{CH}((x_{1,0},\ldots,x_{n,0}),(x_{1,1},\ldots,x_{n,1}))}$:

1 $kp \leftarrow\!\$\ \mathsf{KS.SU}[\mathrm{RO}(3,\cdot)]$
2 $(\mathbf{x}_0, d_0) \leftarrow\!\$\ \mathsf{KS.Enc}[\mathrm{RO}(3,\cdot)](kp, x_{1,0}, \mathrm{RO}(1,(kp,x_{1,0}))),\ldots,(x_{n,0},\mathrm{RO}(1,(kp,x_{n,0})))$
3 $(\mathbf{x}_1, d_1) \leftarrow\!\$\ \mathsf{KS.Enc}[\mathrm{RO}(3,\cdot)](kp,(x_{1,1},\mathrm{RO}(1,(kp,x_{1,1}))),\ldots,(x_{n,1},\mathrm{RO}(1,(kp,x_{n,1}))))$
4 $s \leftarrow\!\$\ \mathbb{F}\ ;\ \mathbf{q} \leftarrow\!\$\ \mathbb{F}^{\mathsf{rel}(n)}\ ;\ (\tau,\omega) \leftarrow \textsc{Run}(\mathbf{x}_0,\mathbf{x}_1,(\mathbf{q},s))$
5 $K \leftarrow (kp,\mathbf{q},s,n)$
6 Return $(K, kp\|\tau, (\omega,\mathbf{q},s))$

Let Rng be the range of functions in $\mathsf{KS.OS}$. A_{v} defines subroutine RO directly using lazy sampling as follows:

$\underline{\mathrm{RO}(a,X)}$:

7 If $\mathrm{R}[a,X] \neq \perp$ then return $\mathrm{R}[a,X]$
8 If $a = 1$ then $\mathrm{R}[a,X] \leftarrow\!\$\ \mathbb{F}$
9 Else If $a = 2$ then $\mathrm{R}[a,X] \leftarrow\!\$\ \{0,1\}^{\ell}$
10 Else $\mathrm{R}[a,X] \leftarrow\!\$\ \mathsf{Rng}$
11 Return $\mathrm{R}[a,X]$

We assume wlog that the adversary queries with correct inputs and so the values in inputs $(x_{1,0},\ldots,x_{n,0})$ are all distinct (same for $(x_{1,1},\ldots,x_{n,1})$). In the CH subroutine, A_{v} first encodes inputs $(x_{1,0},\ldots,x_{n,0})$ and $(x_{1,1},\ldots,x_{n,1})$ using KS to get $\mathbf{x}_0$ and $\mathbf{x}_1$ respectively. For encoding the two inputs, same setup parameter kp is used. This is fine since the setup algorithm is independent of inputs. Note that KS is assumed to be perfectly correct, so $d_0 = d_1 = 1$ always. It then picks s and $\mathbf{q}$ as in Π. Using these values, A_{v} makes its RUN oracle query in line 4. Observe that the output from $\mathsf{F}^{\text{vole}}_{\mathbb{F}}$ for party 2 (adversary here) is ε which makes the functionality output equality check in RUN oracle of InI game always pass. Finally, A_{v} constructs the key K, same as in Π, in line 5 and returns the required output. The challenge bit for A_{v} directly determines the challenge bit for A_{jit}. Since A_{v} returns the bit output by A_{jit}, we get the result as claimed in Eq. (33).

For resource usage, see that $\mathrm{Q}^{\textsc{Run}}(\mathrm{A}_{\text{v}}) = \mathrm{Q}^{\mathrm{CH}}(\mathrm{A}_{\text{jit}})$ as A_{v} makes a RUN oracle query in each CH query from A_{jit}. Note that A_{v} does not have a random oracle. ∎

Games G_0, G_1

INITIALIZE:
1 $\mathsf{G} \leftarrow\!\!\$\ \mathsf{KS.OS}$

TR$(x_1, \ldots, x_n)$:
2 $r \leftarrow r+1$; $n_r \leftarrow n$; $s_{r,0}, s_{r,1} \leftarrow\!\!\$\ \mathbb{F}$; $m \leftarrow \mathsf{el}(n)$
3 $kp_r \leftarrow\!\!\$\ \mathsf{KS.SU}[\mathsf{G}]$
4 $(\mathbf{x}_r, d_r) \leftarrow\!\!\$\ \mathsf{KS.Enc}[\mathsf{G}](kp_r, ((x_1, \mathrm{RO}(1, (kp_r, x_1))), \ldots, (x_n, \mathrm{RO}(1, (kp_r, x_n)))))$
5 $\mathbf{t}_r \leftarrow\!\!\$\ \mathbb{F}^m$; $\mathbf{q}_{r,1} \leftarrow \mathbf{t}_r - \mathbf{x}_r \cdot s_{r,1}$; $\mathbf{q}_{r,0} \leftarrow \mathbf{t}_r - \mathbf{x}_r \cdot s_{r,0}$ // $\mathbf{q}_{r,1} + \mathbf{x}_r \cdot s_{r,1} = \mathbf{q}_{r,0} + \mathbf{x}_r \cdot s_{r,0} = \mathbf{t}_r$
6 $\omega_1, \omega_2 \leftarrow\!\!\$\ \Omega$
7 $(\tau, st_1, st_2) \leftarrow\!\!\$\ \mathbf{Int}_{\Pi^{\mathrm{v}}}(\mathbf{x}_r, (\mathbf{q}_{r,0}, s_{r,0}); \omega_1, \omega_2)$ // Game G_0 only
8 $(\tau, st_1, st_2) \leftarrow\!\!\$\ \mathbf{Int}_{\Pi^{\mathrm{v}}}(\mathbf{x}_r, (\mathbf{q}_{r,1}, s_{r,1}); \omega_1, \omega_2)$ // Game G_1 only
9 For $i = 1, \ldots, n$ do $y_i \leftarrow \langle \mathbf{t}_r, \mathsf{e}[\mathsf{G}](kp_r, x_i, n) \rangle$; $\mathbf{Y}_r[i] \leftarrow \mathrm{RO}(2, (x_i, y_i))$; $\mathrm{T}[r, x_i] \leftarrow \mathbf{Y}_r[i]$
10 Return $(\mathbf{Y}_r, kp_r \| \tau, \omega_1)$

CH$(r', 1, x)$:
11 If not $(r' \le r)$ then return $\perp$
12 If $\mathrm{T}[r', x] \neq \perp$ then return $\mathrm{T}[r', x]$
13 $y \leftarrow \langle \mathbf{t}_{r'}, \mathsf{e}[\mathsf{G}](kp_{r'}, x, n_{r'}) \rangle + (\mathrm{RO}(1, (kp_{r'}, x)) - \langle \mathbf{x}_{r'}, \mathsf{e}[\mathsf{G}](kp_{r'}, x, n_{r'}) \rangle) \cdot s_{r',1}$ // $y = \langle \mathbf{q}_{r',1}, \mathsf{e}[\mathsf{G}](kp_{r'}, x, n_{r'}) \rangle + \mathrm{RO}(1, (kp_{r'}, x)) \cdot s_{r',1}$
14 $\mathrm{T}[r', x] \leftarrow \mathrm{RO}(2, (x, y))$
15 Return $\mathrm{T}[r', x]$

RO(a, X):
16 If $\mathrm{R}[a, X] \neq \perp$ then return $\mathrm{R}[a, X]$
17 If $a = 1$ then $\mathrm{R}[a, X] \leftarrow\!\!\$\ \mathbb{F}$
18 Else If $a = 2$ then $\mathrm{R}[a, X] \leftarrow\!\!\$\ \{0,1\}^{\ell}$
19 Else $\mathrm{R}[a, X] \leftarrow \mathsf{G}(X)$
20 Return $\mathrm{R}[a, X]$

FINALIZE(c'):
21 Return $[[c' = 1]]$

Figure 22: Games for the proof of part (2) of Theorem 8.1.

8.4 Proof for Theorem 8.1 Part (2)

Proof of Theorem 8.1, Part (2): We use a sequence of games $\mathrm{G}_0, \ldots, \mathrm{G}_6$ that will be executed with $\mathrm{A}_{\mathrm{jit}}$. For brevity it is useful to let

$$\epsilon_i = \Pr[\mathrm{G}_i(\mathrm{A}_{\mathrm{jit}})] \ .$$

Games $\mathrm{G}_0, \mathrm{G}_1$ of Figure 22 start from game $\mathbf{G}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi}$ of Figure 5. They then detail the choices and steps in line with the descriptions of f and Π in Figure 21. We assume wlog that in a TR$(x_1, \ldots, x_n)$ query the inputs $x_1, \ldots, x_n$ are all distinct as needed by Π. This will be true for all games going forward. Note that picking $\mathbf{t}_r$ first and then defining $\mathbf{q}_{r,1}, \mathbf{q}_{r,0}$ in terms of $\mathbf{t}_r$ is equivalent to doing it the other way around as in Π. The games are the same except that line 7 is only in G_0 and line 8 is only in G_1. The difference between these two lines is only in the input for party 2 in the call for Π^{v}, this input being $(\mathbf{q}_{r,0}, s_{r,0})$ for G_0 and $(\mathbf{q}_{r,1}, s_{r,1})$ for G_1. Due to line 5 the output $\mathbf{q}_{r,0} + \mathbf{x}_r \cdot s_{r,0}$ of party 1 in the first case is the same as its output $\mathbf{t}_r = \mathbf{q}_{r,1} + \mathbf{x}_r \cdot s_{r,1}$ in the second case. Other than at these lines, both games consistently use $(\mathbf{q}_{r,1}, s_{r,1})$ as the server input for the VOLE. The result is that G_1 is equivalent to the $c = 1$ (i.e., "real") case of $\mathbf{G}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi}$.

We now build adversary A_{v} such that

$$\epsilon_1 - \epsilon_0 \le \mathbf{Adv}^{\mathrm{ini}}_{\mathsf{F}^{\mathrm{vole}}_{\mathbb{F}},\Pi^{\mathrm{v}},2}(\mathrm{A}_{\mathrm{v}}) \ . \tag{36}$$

Adversary A_{v} initializes $r \leftarrow 0$ and runs $c' \leftarrow\!\!\$\ \mathrm{A}_{\mathrm{jit}}[\mathrm{TR}, \mathrm{CH}, \mathrm{RO}]$. It sets $b' \leftarrow c'$ and returns b' as its own output. Here TR, CH, RO are subroutines that A_{v} defines to replace the oracles of the same names that $\mathrm{A}_{\mathrm{jit}}$ expects. In defining the first, A_{v} uses its own RUN oracle, as follows:

TR($x_1, \ldots, x_n$):

$r \leftarrow r+1$; $n_r \leftarrow n$; $s_{r,0}, s_{r,1} \leftarrow\!\!\$\ \mathbb{F}$; $m \leftarrow \mathsf{el}(n)$
$kp_r \leftarrow\!\!\$\ \mathsf{KS.SU}[\mathrm{RO}(3,\cdot)]$
$(\mathbf{x}_r, d_r) \leftarrow\!\!\$\ \mathsf{KS.Enc}[\mathrm{RO}(3,\cdot)](kp_r, ((x_1, \mathrm{RO}(1,(kp_r,x_1))), \ldots, (x_n, \mathrm{RO}(1,(kp_r,x_n)))))$
$\mathbf{t}_r \leftarrow\!\!\$\ \mathbb{F}^m$; $\mathbf{q}_{r,1} \leftarrow \mathbf{t}_r - \mathbf{x}_r \cdot s_{r,1}$; $\mathbf{q}_{r,0} \leftarrow \mathbf{t}_r - \mathbf{x}_r \cdot s_{r,0}$ // $\mathbf{q}_{r,1} + \mathbf{x}_r \cdot s_{r,1} = \mathbf{q}_{r,0} + \mathbf{x}_r \cdot s_{r,0} = \mathbf{t}_r$
$(\tau, \omega_1) \leftarrow \textsc{Run}((\mathbf{q}_{r,0}, s_{r,0}), (\mathbf{q}_{r,1}, s_{r,1}), \mathbf{x}_r)$
For $i = 1, \ldots, n$ do $y_i \leftarrow \langle \mathbf{t}_r, \mathsf{e}[\mathrm{RO}(3,\cdot)](kp_r, x_i, n)\rangle$; $\mathbf{Y}_r[i] \leftarrow \mathrm{RO}(2,(x_i,y_i))$; $\mathrm{T}[r,x_i] \leftarrow \mathbf{Y}_r[i]$
Return $(\mathbf{Y}_r, kp_r \| \tau, \omega_1)$

Above, $\mathbf{q}_{r,1} + \mathbf{x}_r \cdot s_{r,1} = \mathbf{q}_{r,0} + \mathbf{x}_r \cdot s_{r,0}$ ensures that the output of the $\mathsf{F}_{\mathbb{F}}^{\mathrm{vole}}$ for party 1 is the same for the two choices of input given for party 2. Subroutine CH is defined exactly as in games $\mathrm{G}_0, \mathrm{G}_1$, and we do not repeat the code here. The subroutine RO is also similarly defined as in $\mathrm{G}_0, \mathrm{G}_1$ except that instead of using the picked random oracle G in line 19, it lazily samples values from the range of G. This is because picking G upfront is not efficient. (Note that $\mathrm{A_v}$ does not have a random oracle. Its calls to RO, above and in its CH subroutine, refer to calls to the RO subroutine it itself defines.) Let b be the challenge bit in game $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F}_{\mathbb{F}}^{\mathrm{vole}},\Pi^{\mathrm{v}},2}$. Then, in the execution of $\mathrm{A_{jit}}$ by $\mathrm{A_v}$, the former is seeing the same oracle responses that it would get from G_b. Hence

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{ini}}_{\mathsf{F}_{\mathbb{F}}^{\mathrm{vole}},\Pi^{\mathrm{v}},2}(\mathrm{A_v}) &= \Pr\left[\, b' = 1 \mid b = 1 \,\right] - \Pr\left[\, b' = 1 \mid b = 0 \,\right] \\
&= \Pr\left[\, c' = 1 \mid b = 1 \,\right] - \Pr\left[\, c' = 1 \mid b = 0 \,\right] \\
&= \Pr[\mathrm{G}_1(\mathrm{A_{jit}})] - \Pr[\mathrm{G}_0(\mathrm{A_{jit}})] = \epsilon_1 - \epsilon_0 \ ,
\end{aligned}$$

which justifies Eq. (36).

The rest of the proof is devoted to bounding ϵ_0. This is the more difficult part.

Consider games $\mathrm{G}_2, \mathrm{G}_3$ in Figure 23, where G_2 includes the boxed code and G_3 does not. For brevity it is useful to let

$$p(u,x) = \langle \mathbf{x}_u, \mathsf{e}[\mathsf{G}](kp_u, x, n_u)\rangle$$

and

$$\Delta(u,x) = \langle \mathbf{t}_u, \mathsf{e}[\mathsf{G}](kp_u, x, n_u)\rangle + (\mathrm{RO}(1,(kp_u,x)) - p(u,x)) \cdot s_{u,1} \ ,$$

where G is defined at line 1. The function Δ is computed (explicitly) at lines 10, 13, 20 of the games. Our first claim is that G_2 is equivalent to G_0, meaning

$$\epsilon_0 = \epsilon_2 \ . \tag{37}$$

Let us explain the games and justify Eq. (37). Oracle TR in G_2 is equivalent to its counterpart in G_0. In CH, the reply $\mathrm{T}[r',x]$ is, optimistically, chosen at random at line 11. This however could be incorrect, for several reasons. The first, tested at line 12, is that $\mathrm{RO}(2,(x,y))$ has already been queried, in which case, in G_0, it would be the case that $\mathrm{T}[r',x]$ equals $\mathrm{R}[2,(x,y)]$. This is ensured in G_2 by the boxed code in line 12. The second, more subtle reason the optimistic choice of line 11 could be incorrect is that the y from line 10 had appeared previously in a CH or TR query, meaning it was equal to $\Delta(u,x)$ for some $u \neq r'$. Indeed, in that case, in G_0, it would be the case that $\mathrm{T}[r',x]$ would equal $\mathrm{T}[u,x]$. (This is true even if no $\mathrm{RO}(2,\cdot)$ queries have been made so far.) Line 13 tests for such a collision. If it occurs, line 14 sets bad to true, and the boxed code then corrects the optimistic choice of $\mathrm{T}[r',x]$. At line 13 it could be that there are multiple elements u satisfying the condition, so which do we pick to use in the boxed code at line 14? It does not

Games $\boxed{\mathrm{G}_2}$, G_3

INITIALIZE:
1 $\mathsf{G} \leftarrow\!\!\$\ \mathsf{KS.OS}$

TR($x_1, \ldots, x_n$):
2 $r \leftarrow r+1$; $n_r \leftarrow n$; $s_{r,0}, s_{r,1} \leftarrow\!\!\$\ \mathbb{F}$; $m \leftarrow \mathsf{el}(n)$
3 $kp_r \leftarrow\!\!\$\ \mathsf{KS.SU}[\mathsf{G}]$; $(\mathbf{x}_r, d_r) \leftarrow\!\!\$\ \mathsf{KS.Enc}[\mathsf{G}](kp_r, ((x_1, \mathrm{RO}(1,(kp_r,x_1))), \ldots, (x_n, \mathrm{RO}(1,(kp_r,x_n)))))$
4 $\mathbf{t}_r \leftarrow\!\!\$\ \mathbb{F}^m$; $\mathbf{q}_{r,0} \leftarrow \mathbf{t}_r - \mathbf{x}_r \cdot s_{r,0}$
5 $\omega_1, \omega_2 \leftarrow\!\!\$\ \Omega$; $(\tau, st_1, st_2) \leftarrow\!\!\$\ \mathbf{Int}_{\Pi^{\mathsf{v}}}(\mathbf{x}_r, (\mathbf{q}_{r,0}, s_{r,0}); \omega_1, \omega_2)$
6 For $i = 1, \ldots, n$ do $y_i \leftarrow \langle \mathbf{t}_r, \mathsf{e}[\mathsf{G}](kp_r, x_i, n)\rangle$; $\mathbf{Y}_r[i] \leftarrow \mathrm{RO}(2,(x_i,y_i))$; $\mathrm{T}[r,x_i] \leftarrow \mathbf{Y}_r[i]$
7 Return $(\mathbf{Y}_r, kp_r \| \tau, \omega_1)$

CH($r', 1, x$):
8 If not $(r' \le r)$ then return $\perp$
9 If $\mathrm{T}[r',x] \neq \perp$ then return $\mathrm{T}[r',x]$
10 $v \leftarrow \mathsf{e}[\mathsf{G}](kp_{r'}, x, n_{r'})$; $y \leftarrow \langle \mathbf{t}_{r'}, v\rangle + (\mathrm{RO}(1,(kp_{r'},x)) - \langle \mathbf{x}_{r'}, v\rangle) \cdot s_{r',1}$
11 $\mathrm{T}[r',x] \leftarrow\!\!\$\ \{0,1\}^\ell$
12 If $(\mathrm{R}[2,(x,y)] \neq \perp)$ then $\mathsf{bad} \leftarrow 1$; $\boxed{\mathrm{T}[r',x] \leftarrow \mathrm{R}[2,(x,y)]}$
13 If $(\exists u\ :\ u \neq r'$ and $\mathrm{T}[u,x] \neq \perp$ and $y = \langle \mathbf{t}_u, \mathsf{e}[\mathsf{G}](kp_u, x, n_u)\rangle + (\mathrm{RO}(1,(kp_u,x)) - \langle \mathbf{x}_u, \mathsf{e}[\mathsf{G}](kp_u,x,n_u)\rangle) \cdot s_{u,1})$ then
14 $\quad \mathsf{bad} \leftarrow 1$; $\boxed{\mathrm{T}[r',x] \leftarrow \mathrm{T}[u,x]}$
15 Return $\mathrm{T}[r',x]$

RO(a, X):
16 If $\mathrm{R}[a,X] \neq \perp$ then return $\mathrm{R}[a,X]$
17 If $a = 1$ then $\mathrm{R}[a,X] \leftarrow\!\!\$\ \mathbb{F}$
18 Else If $a = 2$ then
19 $\quad (x,y) \leftarrow X$; $\mathrm{R}[a,X] \leftarrow\!\!\$\ \{0,1\}^\ell$
20 $\quad$ If $(\exists u\ :\ \mathrm{T}[u,x] \neq \perp$ and $y = \langle \mathbf{t}_u, \mathsf{e}[\mathsf{G}](kp_u,x,n_u)\rangle + (\mathrm{RO}(1,(kp_u,x)) - \langle \mathbf{x}_u, \mathsf{e}[\mathsf{G}](kp_u,x,n_u)\rangle) \cdot s_{u,1})$ then
21 $\quad\quad \mathsf{bad} \leftarrow 1$; $\boxed{\mathrm{R}[a,X] \leftarrow \mathrm{T}[u,x]}$
22 Else $\mathrm{R}[a,X] \leftarrow \mathsf{G}(X)$
23 Return $\mathrm{R}[a,X]$

FINALIZE(c'):
24 Return $[[c' = 1]]$

Figure 23: More games for the proof of part (2) of Theorem 8.1. The boxed code is included only in G_2.

matter, because the T-values of those choices would be the same. We now discuss RO. In the $a = 1$ case it is the same as in G_0. In the $a = 2$ case, line 19 parses X as (x, y) and optimistically picks the reply $\mathrm{R}[a, X]$ at random. This is however incorrect if y already occurred as a potential input for $\mathrm{RO}[2, (x, \cdot)]$ in a CH query, meaning if the condition tested at line 20 is true, because in that case, in G_0, the values $\mathrm{R}[a, X]$ and $\mathrm{T}[u, x]$ would be the same. Line 21 makes the corresponding correction. Again at line 20 there could be many choices of u but due to lines 12–14, they would yield the same result for line 21. This concludes our justification of Eq. (37).

Games $\mathrm{G}_2, \mathrm{G}_3$ are identical-until-bad as defined in [9]. By the Fundamental Lemma of Game Playing in [9] (Lemma 2.1), we thus have

$$\epsilon_2 - \epsilon_3 \le \Pr[\mathrm{G}_3(\mathrm{A}_{\mathsf{jit}}) \text{ sets } \mathsf{bad}] \ . \tag{38}$$

Recall that G_3 does not contain the boxed code. Thus, even though various conditions are tested at lines 12, 13, 20, they do not change the optimistic choices of lines 11, 19, meaning those are what the respective oracles return. The result is that G_3 is equivalent to the $c = 0$ ("random") case of game $\mathbf{G}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi}$. Note this is true even though the transcript τ at line 5 is generated using

Games G_4, $\boxed{\mathrm{G}_5}$

INITIALIZE:
1 $\mathsf{G} \leftarrow\!\!\$\ \mathsf{KS.OS}$

TR$(x_1, \ldots, x_n)$:
2 $r \leftarrow r+1$; $n_r \leftarrow n$; $s_{r,0}, s_{r,1} \leftarrow\!\!\$\ \mathbb{F}$; $m \leftarrow \mathsf{el}(n)$; $S_r \leftarrow \{x_1, \ldots, x_n\}$
3 If $(s_{r,1} = 0)$ then $\mathsf{bad} \leftarrow 1$; $\boxed{s_{r,1} \leftarrow\!\!\$\ \mathbb{F} \setminus \{0\}}$
4 $kp_r \leftarrow\!\!\$\ \mathsf{KS.SU}[\mathsf{G}]$; $(\mathbf{x}_r, d_r) \leftarrow\!\!\$\ \mathsf{KS.Enc}[\mathsf{G}](kp_r, ((x_1, \mathrm{RO}(1, (kp_r, x_1))), \ldots, (x_n, \mathrm{RO}(1, (kp_r, x_n)))))$
5 If $(\exists x\ :\ x \notin S_r$ and $\mathrm{R}[1, (\mathrm{kp_r, x})] = \langle \mathbf{x}_\mathrm{r}, \mathsf{e}[\mathsf{G}](\mathrm{kp_r, x, n_r}) \rangle)$ then $\mathsf{bad} \leftarrow 1$; $\boxed{r \leftarrow r-1 \text{ ; return } \perp}$
6 $\mathbf{t}_r \leftarrow\!\!\$\ \mathbb{F}^m$; $\mathbf{q}_{r,0} \leftarrow \mathbf{t}_r - \mathbf{x}_r \cdot s_{r,0}$
7 $\omega_1, \omega_2 \leftarrow\!\!\$\ \Omega$; $(\tau, st_1, st_2) \leftarrow\!\!\$\ \mathbf{Int}_{\Pi^\mathrm{v}}(\mathbf{x}_r, (\mathbf{q}_{r,0}, s_{r,0}); \omega_1, \omega_2)$
8 For $i = 1, \ldots, n$ do $y_i \leftarrow \langle \mathbf{t}_r, \mathsf{e}[\mathsf{G}](kp_r, x_i, n) \rangle$; $\mathbf{Y}_r[i] \leftarrow \mathrm{RO}(2, (x_i, y_i))$; $\mathrm{T}[r, x_i] \leftarrow \mathbf{Y}_r[i]$
9 Return $(\mathbf{Y}_r, kp_r \| \tau, \omega_1)$

CH$(r', 1, x)$:
10 If not $(r' \le r)$ then return $\perp$
11 If $\mathrm{T}[r', x] \neq \perp$ then return $\mathrm{T}[r', x]$
12 $v \leftarrow \mathsf{e}[\mathsf{G}](kp_{r'}, x, n_{r'})$; $y \leftarrow \langle \mathbf{t}_{r'}, v \rangle + (\mathrm{RO}(1, (kp_{r'}, x)) - \langle \mathbf{x}_{r'}, v \rangle) \cdot s_{r',1}$
13 $\mathrm{T}[r', x] \leftarrow\!\!\$\ \{0,1\}^\ell$
14 If $(\mathrm{R}[2, (x, y)] \neq \perp)$ then $\mathsf{win} \leftarrow 1$
15 If $(\exists u\ :\ u \neq r'$ and $\mathrm{T}[u, x] \neq \perp$ and $y = \langle \mathbf{t}_u, \mathsf{e}[\mathsf{G}](kp_u, x, n_u) \rangle + (\mathrm{RO}(1, (kp_u, x) - \langle \mathbf{x}_u, \mathsf{e}[\mathsf{G}](kp_u, x, n_u) \rangle) \cdot s_{u,1})$ then
16 $\quad \mathsf{win} \leftarrow 1$
17 Return $\mathrm{T}[r', x]$

RO(a, X):
18 If $\mathrm{R}[a, X] \neq \perp$ then return $\mathrm{R}[a, X]$
19 If $a = 1$ then $\mathrm{R}[a, X] \leftarrow\!\!\$\ \mathbb{F}$
20 Else If $a = 2$ then
21 $\quad (x, y) \leftarrow X$; $\mathrm{R}[a, X] \leftarrow\!\!\$\ \{0,1\}^\ell$
22 $\quad$ If $(\exists u\ :\ \mathrm{T}[u, x] \neq \perp$ and $y = \langle \mathbf{t}_u, \mathsf{e}[\mathsf{G}](kp_u, x, n_u) \rangle + (\mathrm{RO}(1, (kp_u, x) - \langle \mathbf{x}_u, \mathsf{e}[\mathsf{G}](kp_u, x, n_u) \rangle) \cdot s_{u,1})$ then
23 $\quad\quad \mathsf{win} \leftarrow 1$
24 Else $\mathrm{R}[a, X] \leftarrow \mathsf{G}(X)$
25 Return $\mathrm{R}[a, X]$

FINALIZE(c'):
26 Return win

Figure 24: More games for the proof of part (2) of Theorem 8.1. The boxed code is included only in G_5.

$s_{r,0}$, meaning we are seeing $(\mathbf{q}_{r,0}, s_{r,0})$ as the server VOLE inputs for this claim. Thus we have

$$\begin{aligned}
&\mathbf{Adv}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi}(\mathrm{A}_{\mathrm{jit}}) = \epsilon_1 - \epsilon_3 \\
&= (\epsilon_1 - \epsilon_0) + (\epsilon_0 - \epsilon_2) + (\epsilon_2 - \epsilon_3) \\
&\le \mathbf{Adv}^{\mathrm{ini}}_{\mathsf{F}^{\mathrm{vole}}_{\mathbb{F}}, \Pi^\mathrm{v}, 1}(\mathrm{A}_\mathrm{v}) + \Pr[\mathrm{G}_3(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{bad}] \ . \qquad (39)
\end{aligned}$$

To bound $\Pr[\mathrm{G}_3(\mathrm{A}_{\mathrm{jit}})$ sets $\mathsf{bad}]$, consider games $\mathrm{G}_4, \mathrm{G}_5$ of Figure 24, where G_5 contains the boxed code and G_4 does not. Where G_3 sets bad, games $\mathrm{G}_4, \mathrm{G}_5$ set win, and FINALIZE returns win. Lines 3, 5 in G_4 can be ignored, since they do not change what the game returns. Thus we have

$$\Pr[\mathrm{G}_3(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{bad}] = \epsilon_4 \ .$$

Games $\mathrm{G}_4, \mathrm{G}_5$ are identical-until-bad as defined in [9]. By the Fundamental Lemma of Game Playing in [9] (Lemma 2.1), we thus have

$$\epsilon_4 = \epsilon_5 + (\epsilon_4 - \epsilon_5) \le \epsilon_5 + \Pr[\mathrm{G}_4(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{bad}] \ . \qquad (40)$$

We now have to bound the two terms above. We start with the second, claiming first that

$$\Pr[\mathrm{G}_4(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{bad} \text{ at line 3}] \leq \frac{Q_T}{|\mathbb{F}|} \ . \tag{41}$$

This is straightforward since the probability that line 3 sets bad is $1/|\mathbb{F}|$, so this line contributes at most $Q_T/|\mathbb{F}|$ overall. Next, we build an adversary A_{en} such that

$$\Pr[\mathrm{G}_4(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{bad} \text{ at line 5}] \leq \mathbf{Adv}^{\mathrm{enuf}}_{\mathsf{KS}}(\mathrm{A}_{\mathrm{en}}) \ . \tag{42}$$

Adversary A_{en} initializes $r \leftarrow 0$ and runs $c' \leftarrow \mathrm{A}_{\mathrm{jit}}[\mathrm{TR}, \mathrm{CH}, \mathrm{RO}]$. Here $\mathrm{TR}, \mathrm{CH}, \mathrm{RO}$ are subroutines that A_{v} defines to replace the oracles of the same names that $\mathrm{A}_{\mathrm{jit}}$ expects. Subroutine RO is defined as follows:

$\underline{\mathrm{RO}(a, X)}$:

1 If $\mathrm{R}[a, X] \neq \bot$ then return $\mathrm{R}[a, X]$
2 If $a = 1$ then $(kp, x) \leftarrow X$; $S_1 \leftarrow S_1 \cup \{x\}$; $\mathrm{R}[a, X] \leftarrow\!\!\$\ \textsc{Value}(X)$
3 Else If $a = 2$ then
4 $\quad (x, y) \leftarrow X$; $\mathrm{R}[a, X] \leftarrow\!\!\$\ \{0,1\}^{\ell}$
5 $\quad$ If $(\exists\, u\ :\ \mathrm{T}[u, x] \neq \bot$ and $y = \langle \mathbf{t}_u, \mathsf{e}[\mathsf{G}](kp_u, x, n_u)\rangle + (\mathrm{RO}(1, (kp_u, x) - \langle \mathbf{x}_u, \mathsf{e}[\mathsf{G}](kp_u, x, n_u)\rangle)) \cdot s_{u,1})$ then
6 $\quad\quad \mathsf{win} \leftarrow 1$
7 Else $\mathrm{R}[a, X] \leftarrow \mathsf{G}(X)$
8 Return $\mathrm{R}[a, X]$

The only change in the subroutine above from the code in RO oracle of G_4 is that in the $a = 1$ case, the table entry $\mathrm{R}(a, X)$ is filled using the output of $\textsc{Value}(X)$. Subroutine TR is defined as follows:

$\underline{\mathrm{TR}(x_1, \ldots, x_n)}$:

1 $r \leftarrow r + 1$; $n_r \leftarrow n$; $s_{r,0}, s_{r,1} \leftarrow\!\!\$\ \mathbb{F}$; $m \leftarrow \mathsf{el}(n)$; $S_r \leftarrow \{x_1, \ldots, x_n\}$
2 $(w, (\omega, \omega')) \leftarrow \textsc{Vf}((x_1, \ldots, x_n), S_1)$
3 $kp_r \leftarrow \mathsf{KS.SU}[\mathsf{G}](;\omega)$; $(\mathbf{x}_r, d_r) \leftarrow \mathsf{KS.Enc}[\mathsf{G}](kp_r, ((x_1, \mathrm{RO}(1, (kp_r, x_1))), \ldots, (x_n, \mathrm{RO}(1, (kp_r, x_n)))); \omega')$
4 $\mathbf{t}_r \leftarrow\!\!\$\ \mathbb{F}^m$; $\mathbf{q}_{r,0} \leftarrow \mathbf{t}_r - \mathbf{x}_r \cdot s_{r,0}$
5 $\omega_1, \omega_2 \leftarrow\!\!\$\ \Omega$; $(\tau, st_1, st_2) \leftarrow\!\!\$\ \mathbf{Int}_{\Pi^{\mathrm{v}}}(\mathbf{x}_r, (\mathbf{q}_{r,0}, s_{r,0}); \omega_1, \omega_2)$
6 For $i = 1, \ldots, n$ do $y_i \leftarrow \langle \mathbf{t}_r, \mathsf{e}[\mathsf{G}](kp_r, x_i, n)\rangle$; $\mathbf{Y}_r[i] \leftarrow \mathrm{RO}(2, (x_i, y_i))$; $\mathrm{T}[r, x_i] \leftarrow \mathbf{Y}_r[i]$
7 Return $(\mathbf{Y}_r, kp_r \| \tau, \omega_1)$

Subroutine CH is the same as in G_4 and so we do not repeat its code here.

First, observe that $\textsc{Value}(X)$ also lazily samples a value uniformly from $\mathbb{F}$ as done in line 19 of RO of G_4. In the TR subroutine, lines 4,5 in TR oracle of G_4 are replaced with $\textsc{Vf}$ call in line 2 above. The coins output from this are then used to create the required values (setup parameter kp_r and encoding output $\mathbf{x}_r, d_r$) for the rest of the subroutine, which has the same code as in G_4.TR. Due to this and the fact that $\mathrm{RO}(1, X) = \textsc{Value}(X)$, we know that $kp_r, \mathbf{x}_r$ are the values used in $\textsc{Vf}$ oracle to make the required check. This makes A_{en} correctly simulate the oracles that $\mathrm{A}_{\mathrm{jit}}$ expects.

We know that the encoding always passes since KS is perfectly correct, so the first check in $\textsc{Vf}$ oracle always passes. With $\mathrm{RO}(1, X) = \textsc{Value}(X)$ and because S_1 is used in the $\textsc{Vf}$ oracle call we can see that the condition checked in line 5 of G_4.TR is same as the condition checked in $\textsc{Vf}$ oracle. Therefore, if $\mathrm{A}_{\mathrm{jit}}$ sets bad in G_4, A_{en} would make a $\textsc{Vf}$ query which sets the win flag. Thus, we get Eq. (42).

Games G_6, $\boxed{\mathrm{G}_7}$

INITIALIZE:
1 $\mathsf{G} \leftarrow\!\!\$\ \mathsf{KS.OS}$

TR$(x_1, \ldots, x_n)$:
2 $r \leftarrow r+1$; $n_r \leftarrow n$; $s_{r,0} \leftarrow\!\!\$\ \mathbb{F}$; $s_{r,1} \leftarrow\!\!\$\ \mathbb{F} \setminus \{0\}$; $m \leftarrow \mathsf{el}(n)$; $S_r \leftarrow \{x_1, \ldots, x_n\}$
3 $kp_r \leftarrow\!\!\$\ \mathsf{KS.SU}[\mathsf{G}]$; $(\mathbf{x}_r, d_r) \leftarrow\!\!\$\ \mathsf{KS.}\textsc{Enc}[\mathsf{G}](kp_r, ((x_1, \mathrm{RO}(1, (kp_r, x_1))), \ldots, (x_n, \mathrm{RO}(1, (kp_r, x_n)))))$
4 If $(\exists x\ :\ x \notin S_r$ and $\mathrm{R}[1, (\mathrm{kp_r}, \mathrm{x})] = \langle \mathbf{x}_\mathrm{r}, \mathsf{e}[\mathsf{G}](\mathrm{kp_r}, \mathrm{x}, \mathrm{n_r})\rangle)$ then $r \leftarrow r-1$; return $\perp$
5 $\mathbf{t}_r \leftarrow\!\!\$\ \mathbb{F}^m$; $\mathbf{q}_{r,0} \leftarrow \mathbf{t}_r - \mathbf{x}_r \cdot s_{r,0}$
6 $\omega_1, \omega_2 \leftarrow\!\!\$\ \Omega$; $(\tau, st_1, st_2) \leftarrow\!\!\$\ \mathbf{Int}_{\Pi^\mathrm{v}}(\mathbf{x}_r, (\mathbf{q}_{r,0}, s_{r,0}); \omega_1, \omega_2)$
7 For $i = 1, \ldots, n$ do $y_i \leftarrow \langle \mathbf{t}_r, \mathsf{e}[\mathsf{G}](kp_r, x_i, n)\rangle$; $\mathbf{Y}_r[i] \leftarrow \mathrm{RO}(2, (x_i, y_i))$; $\mathrm{T}[r, x_i] \leftarrow \mathbf{Y}_r[i]$
8 Return $(\mathbf{Y}_r, kp_r \| \tau, \omega_1)$

CH$(r', 1, x)$:
9 If not $(r' \le r)$ then return $\perp$
10 If $\mathrm{T}[r', x] \neq \perp$ then return $\mathrm{T}[r', x]$
11 $v \leftarrow \mathsf{e}[\mathsf{G}](kp_{r'}, x, n_{r'})$; $y \leftarrow \langle \mathbf{t}_{r'}, v\rangle + (\mathrm{RO}(1, (kp_{r'}, x)) - \langle \mathbf{x}_{r'}, v\rangle) \cdot s_{r',1}$
12 $\mathrm{T}[r', x] \leftarrow\!\!\$\ \{0,1\}^\ell$
13 If $(\mathrm{R}[2, (x, y)] \neq \perp)$ then $\mathsf{win} \leftarrow 1$
14 If $(\exists u\ :\ u \neq r'$ and $\mathrm{T}[u, x] \neq \perp$ and $y = \langle \mathbf{t}_u, \mathsf{e}[\mathsf{G}](kp_u, x, n_u)\rangle + (\mathrm{RO}(1, (kp_u, x)) - \langle \mathbf{x}_u, \mathsf{e}[\mathsf{G}](kp_u, x, n_u)\rangle) \cdot s_{u,1})$ then $\mathsf{win} \leftarrow 1$
15 Return $\mathrm{T}[r', x]$

RO(a, X):
16 If $\mathrm{R}[a, X] \neq \perp$ then return $\mathrm{R}[a, X]$
17 If $a = 1$ then
18 $\quad h \leftarrow\!\!\$\ \mathbb{F}$; (kp, x) ; $S \leftarrow \{\ \langle \mathbf{x}_u, \mathsf{e}[\mathsf{G}](kp_u, x, n_u)\rangle\ :\ u \in [1..r]\ \}$
19 $\quad$ If $h \in S$ then $\mathsf{bad} \leftarrow 1$; $\boxed{h \leftarrow\!\!\$\ \mathbb{F} \setminus S}$
20 $\quad \mathrm{R}[a, X] \leftarrow h$
21 Else If $a = 2$ then
22 $\quad (x, y) \leftarrow X$; $\mathrm{R}[a, X] \leftarrow\!\!\$\ \{0,1\}^\ell$
23 $\quad$ If $(\exists u\ :\ \mathrm{T}[u, x] \neq \perp$ and $y = \langle \mathbf{t}_u, \mathsf{e}[\mathsf{G}](kp_u, x, n_u)\rangle + (\mathrm{RO}(1, (kp_u, x)) - \langle \mathbf{x}_u, \mathsf{e}[\mathsf{G}](kp_u, x, n_u)\rangle) \cdot s_{u,1})$ then $\mathsf{win} \leftarrow 1$
24 Else $\mathrm{R}[a, X] \leftarrow \mathsf{G}(X)$
25 Return $\mathrm{R}[a, X]$

FINALIZE(c'):
26 Return win

Figure 25: Games $\mathrm{G}_6, \mathrm{G}_7$ for the proof of part (2) of Theorem 8.1.

Now we have to bound ϵ_5. Towards this, we use games $\mathrm{G}_6, \mathrm{G}_7$ of Figure 25. G_6 rewrites G_5 in simplified form and line 19 in G_6 can be ignored as it does not change the output. We have

$$\epsilon_5 = \epsilon_6\ .$$

Games $\mathrm{G}_6, \mathrm{G}_7$ are identical-until-bad as defined in [9]. By the Fundamental Lemma of Game Playing in [9] (Lemma 2.1), we thus have

$$\epsilon_6 = \epsilon_7 + (\epsilon_6 - \epsilon_7) \le \epsilon_7 + \Pr[\mathrm{G}_6(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{bad}]\ . \tag{43}$$

We now have to bound the two terms above. We start with the second, claiming first that

$$\Pr[\mathrm{G}_6(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{bad}] \le \frac{Q_T \cdot Q_1}{|\mathbb{F}|}\ . \tag{44}$$

The bad flag is set in line 19 in G_6. It is set if the uniformly picked value h is one of elements of set S. This set contains all the field elements obtained by computing $p(u, x)$ for all $u \in [1..r]$. Hence, its size is at most Q_T. Accounting for Q_1 queries of the form $\mathrm{RO}(1, \cdot)$, we get Eq. (44).

We now bound ϵ_7. A key observation towards this is that the adversary has no information about

$s_{1,1}, \ldots, s_{r,1}$. We first claim

$$\Pr[\mathrm{G}_7(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{win} \text{ at line } 13] \leq \frac{Q_c}{|\mathbb{F}| - 1} \tag{45}$$

$$\Pr[\mathrm{G}_7(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{win} \text{ at line } 23] \leq \frac{Q_T \cdot Q_2}{|\mathbb{F}| - 1} \ . \tag{46}$$

Line 13 sets win if $\mathrm{A}_{\mathrm{jit}}$ previously made a query $\mathrm{RO}(2, (x, y))$ where $y \leftarrow \langle \mathbf{t}_{r'}, \mathsf{e}[\mathsf{G}](kp_{r'}, x, n_{r'}) \rangle + (\mathrm{R}[1, (kp_{r'}, x)] - p(r', x)) \cdot s_{r'}$ is computed in line 11. For this, $\mathrm{A}_{\mathrm{jit}}$ would need to compute this y. Observe that at line 13 we will have $x \notin S_{r'}$ due to line 10, so $\mathrm{A}_{\mathrm{jit}}$ does not already know y. The values in first term of y are known to $\mathrm{A}_{\mathrm{jit}}$, so it can be easily computed. The following bounds the probability of adversary computing the second term of y which is $(\mathrm{R}[1, (kp_{r'}, x)] - p(r', x)) \cdot s_{r',1}$. $\mathrm{A}_{\mathrm{jit}}$ can compute $\alpha \leftarrow (\mathrm{R}[1, (kp_{r'}, x)] - p(r', x))$. If $\alpha = 0$ then the second term of y is 0, that is, easily computable by $\mathrm{A}_{\mathrm{jit}}$. But due to the lines 4 and 19 and the fact that $x \notin S_{r'}$, it must be that $\alpha \neq 0$. Above we noted that $\mathrm{A}_{\mathrm{jit}}$ has no information about $s_{r',1}$. So the probability that $\mathrm{A}_{\mathrm{jit}}$ computes $\alpha \cdot s_{r',1}$ is at most $1/(|\mathbb{F}| - 1)$. Accounting for the Q_c queries to CH, this justifies Eq. (45). Similarly at line 19 we will have $x \notin S_u$ since $\mathrm{T}[u, x] \neq \perp$ as per the test on the same line, and thus a similar argument holds. However, here, there are at most Q_T values, instead of 1, that would lead to a favorable outcome (computing a correct y value) for $\mathrm{A}_{\mathrm{jit}}$. Thus, probability that $\mathrm{A}_{\mathrm{jit}}$ sets the win flag in a single $\mathrm{RO}(2, \cdot)$ query is at most $Q_T/(|\mathbb{F}| - 1)$. Accounting for the Q_2 queries to $\mathrm{RO}(2, \cdot)$, this justifies Eq. (46). Now we claim

$$\Pr[\mathrm{G}_7(\mathrm{A}_{\mathrm{jit}}) \text{ sets } \mathsf{win} \text{ at line } 14] \leq \frac{(Q_T - 1) \cdot Q_c}{|\mathbb{F}| - 1} \ . \tag{47}$$

The collision of line 13 occurs if

$$\langle \mathbf{t}_{r'}, p(r', x) \rangle + (\mathrm{R}[1, (kp_{r'}, x)] - p(r', x)) \cdot s_{r',1} = \langle \mathbf{t}_u, p(u, x) \rangle + (\mathrm{R}[1, (kp_u, x)] - p(u, x)) \cdot s_{u,1} \ .$$

We know that $u \neq r'$. Line 10 implies $x \notin S_{r'}$ and so due to lines 4, 19 we have $\mathrm{R}[1, (kp_{r'}, x)] - p(r', x) \neq 0$. Then as above, the choice of $s_{r',1}$ from $\mathbb{F} \setminus \{0\}$, and the fact that the adversary has no information about it, means that the probability of the collision is at most $(Q_T - 1)/(|\mathbb{F}| - 1)$ as there are at most $Q_T - 1$ possible values of u. Accounting for the Q_c queries to CH, this justifies Eq. (47).

Putting this together and using that $|\mathbb{F}| - 1 \geq |\mathbb{F}|/2$ we have

$$\epsilon_7 \leq \frac{Q_T \cdot (Q_c + Q_2)}{|\mathbb{F}| - 1} \leq \frac{2Q_T \cdot (Q_c + Q_2)}{|\mathbb{F}|} \ .$$

Putting all the above together concludes the proof. ▌

The above shows that server security of $\Pi = \mathbf{J3}[\mathsf{KS}, \ell, \Pi^{\mathrm{v}}]$ requires KS to be EnUF secure. One may wonder whether the LKVS constructions in prior works achieve this. While it is our belief that they do, instead of analyzing them individually, in the following, we present a universal transform to convert any LKVS scheme into one that information-theoretically achieves EnUF security with minimal overhead.

8.5 Universally Achieving EnUF Security

TRANSFORM. We describe a transform $\mathbf{KT}$ which builds an LKVS scheme KS^* from any LKVS KS and integer $sl \geq 0$ via $\mathsf{KS}^* \leftarrow \mathbf{KT}[\mathsf{KS}, sl]$. Later we will show that the transformed output KS^* is information-theoretically EnUF secure. The scheme KS^* has the same parameters as KS,

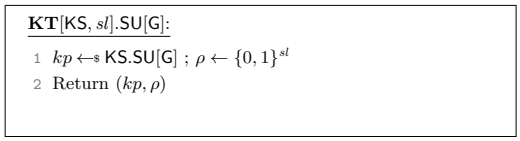


Figure 26: Setup algorithm for $\mathsf{KS}^* \leftarrow \mathbf{KT}[\mathsf{KS}]$.

$\mathsf{KS}^*.\mathbb{F} = \mathsf{KS}.\mathbb{F}$, $\mathsf{KS}^*.\mathsf{OS} = \mathsf{KS}.\mathsf{OS}$. The only change is in the setup algorithm which is shown in Figure 26. Here, it uniformly samples a string (salt) ρ of length sl and returns it in addition to the setup parameter kp output from the setup algorithm of KS. Since setup parameters are input to VALUE in the EnUF game, adding this salt leads to EnUF security irrespective of EnUF security of KS which we will show below. The embedding, encoding and decoding algorithms of KS^* return the output of running the respective algorithms of KS with setup parameter kp. The salt, ρ is ignored in these algorithms.

That correctness condition is satisfied by KS^*, if KS is correct, comes directly since they have the same output for embedding, encoding and decoding algorithms.

It is also straightforward to see that the correctness error of KS^* is bounded by the correctness error of KS since KS^* returns the same decision bit. We omit the proof for this. This means if the underlying scheme KS is perfectly correct then the transformed scheme KS^* is perfectly correct as well.

In the following theorem, we show that KS^* is EnUF always secure as long as parameters, specifically field size and salt length, are selected large enough.

Theorem 8.2 *Let* KS *be an LKVS scheme,* $sl \geq 1$ *and* $\mathsf{KS}^* = \mathbf{KT}[\mathsf{KS}, sl]$. *Let* A_{en} *be any adversary playing the game* $\mathbf{G}^{\mathrm{enuf}}_{\mathsf{KS}^*}$. *Assume the sum of size of set* S *across all* VF *oracle queries of* A_{en} *is* Q. *We have*

$$\mathbf{Adv}^{\mathrm{enuf}}_{\mathsf{KS}^*}(\mathrm{A}_{\mathrm{en}}) \leq Q \cdot \left(\frac{1}{|\mathbb{F}|} + \frac{Q_v}{2^{sl}} \right) . \tag{48}$$

where $Q_v = \mathrm{Q}^{\mathrm{VALUE}}(\mathrm{A}_{\mathrm{en}})$.

Proof of Theorem 8.2: We will use the games $\mathrm{G}_0, \mathrm{G}_1$ shown in Figure 27. First observe that

$$\Pr[\mathrm{G}_0(\mathrm{A}_{\mathrm{en}})] = \mathbf{Adv}^{\mathrm{enuf}}_{\mathsf{KS}^*}(\mathrm{A}_{\mathrm{en}}) . \tag{49}$$

This is because G_0 is simply the game $\mathbf{G}^{\mathrm{enuf}}_{\mathsf{KS}^*}$ with details added. The VALUE and RO oracles have exactly the same code. In the VF oracle, lines 6,7,8 detail the steps of $\mathsf{KS}^*.\mathsf{SU}$ and $\mathsf{KS}^*.\mathsf{Enc}$ algorithms. The condition to set the win flag in $\mathbf{G}^{\mathrm{enuf}}_{\mathsf{KS}^*}.\mathrm{VF}$ is checked here in parts. Line 9 checks the condition for decision bit. Through lines 5,10, G_0 ensures that win can only be set if the key $x \in S$ used in decoding is not one of $x_1, \ldots, x_n$. Finally, G_0 tests $v = \langle \mathbf{x}, \mathsf{KS.e}[\mathrm{RO}](kp, \mathbf{x}, x, n) \rangle$ where v is uniformly sampled from $\mathsf{KS}.\mathbb{F}$. This is the same as testing the equality with $\mathrm{VALUE}((kp^*, x))$ as done in $\mathbf{G}^{\mathrm{enuf}}_{\mathsf{KS}^*}.\mathrm{VF}$ except when the table entry $\mathrm{T}[(kp, \rho), x]$ is already filled from a prior VALUE call. This is checked and fixed in line 12. Thus, we get Eq. (49).

The games $\mathrm{G}_0, \mathrm{G}_1$ are identical-until-bad as defined in [9]. By the Fundamental Lemma of Game Playing in [9] (Lemma 2.1), we thus have

$$\Pr[\mathrm{G}_0(\mathrm{A}_{\mathrm{en}})] \leq \Pr[\mathrm{G}_1(\mathrm{A}_{\mathrm{en}})] + \Pr[\mathrm{G}_1(\mathrm{A}_{\mathrm{en}}) \text{ sets } \mathsf{bad}]$$

We now bound the terms in the right-hand-side of the above equation.

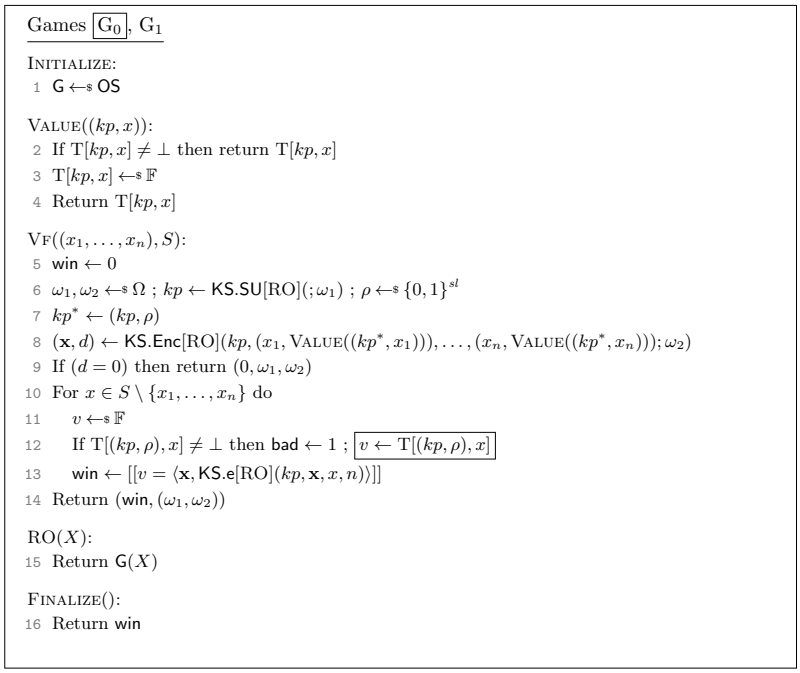


Figure 27: Games for proof of Theorem 8.2.

We claim that

$$\Pr[\mathrm{G}_1(\mathrm{A}_{\mathrm{en}}) \text{ sets } \mathsf{bad}] \leq \frac{Q \cdot Q_v}{2^{sl}} \ . \tag{50}$$

The bad flag in G_1 is set only if A_{en} made a prior $\textsc{Value}((kp, \rho), x)$ query. Since ρ is uniformly selected, in each VF oracle query, the above can only happen with probability at most $(|S| \cdot Q_v)/2^{sl}$. Accounting for size of set S across all VF queries, we get Eq. (50).

Finally, we claim that

$$\Pr[\mathrm{G}_1(\mathrm{A}_{\mathrm{en}})] \leq \frac{Q}{|\mathbb{F}|} \ . \tag{51}$$

The win flag is set in line 13 of G_1 if $v = \langle \mathbf{x}, \mathsf{KS.e}[\mathrm{RO}](kp, \mathbf{x}, x, n)\rangle$. Since v is uniformly selected from $\mathbb{F}$, in a VF oracle query, win will be set with probability at most $|S|/|\mathbb{F}|$. Accounting for all VF oracle queries we get Eq. (51). ∎

Using the construction and results of above, we can now construct, from any perfectly correct LKVS scheme KS and VOLE protocol Π^{v}, a JIT-OPRF protocol $\Pi \leftarrow \mathbf{J3}[\mathsf{KS}^*, \ell, \Pi^{\mathrm{v}}]$ where $\mathsf{KS}^* = \mathbf{KT}[\mathsf{KS}, sl]$ whose server security only relies on the InI (for honest party 2) security of Π^{v}. This means, the computational term for EnUF security in Eq. (34) is substituted with the bound in Eq. (48).

How large should sl be? We would want to make $\mathbf{Adv}^{\mathrm{enuf}}_{kp^*}(\mathrm{A}_{\mathrm{en}})$ very small, for eg, 2^{-33} (as desired in discussions from prior sections). From the proof in section 8.4 and using the values in the concrete parameters discussions of previous sections, we can see that $Q \approx Q_1 \cdot Q_T \approx 2^{70}$ and $Q_v \approx Q_1 \approx 2^{60}$. Then from Eq. (48), to make $\mathbf{Adv}^{\mathrm{enuf}}_{kp^*}(\mathrm{A}_{\mathrm{en}}) \leq 2^{-33}$, we get $sl \geq 164$ and $|\mathbb{F}| \geq 2^{104}$ (to make each individual term in Eq. (48) $\leq 2^{34}$). Note that from the discussion in section 8.2, we already know that $|\mathbb{F}| \geq 2^{103}$ is needed.

Acknowledgments

We thank Roy Stracovsky for participation in the early stages of this work.

References

[1] Apple: Password monitoring. https://support.apple.com/guide/security/password-monitoring-sec78e79fc3b/web (Feb 2021) (Cited on page 3.)

[2] Applebaum, B., Damgård, I., Ishai, Y., Nielsen, M., Zichron, L.: Secure arithmetic computation with constant computational overhead. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 223–254. Springer, Cham (Aug 2017) (Cited on page 3.)

[3] 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 3.)

[4] Bellare, M., Davis, H., Günther, F.: Separate your domains: NIST PQC KEMs, oracle cloning and read-only indifferentiability. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part II. LNCS, vol. 12106, pp. 3–32. Springer, Cham (May 2020) (Cited on page 21, 32.)

[5] Bellare, M., Micali, S.: Non-interactive oblivious transfer and applications. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 547–557. Springer, New York (Aug 1990) (Cited on page 21, 23.)

[6] Bellare, M., Ranjan, R., Riepel, D.: Just-in-time-oprfs and a modular framework for fast private set intersection. Crypto 2026 (Cited on page 9, 23, 33, 34.)

[7] Bellare, M., Ranjan, R., Riepel, D., Aldakheel, A.: The concrete security of two-party computation: Simple definitions, and tight proofs for PSI and OPRFs. In: Chung, K.M., Sasaki, Y. (eds.) ASIACRYPT 2024, Part VI. LNCS, vol. 15489, pp. 328–362. Springer, Singapore (Dec 2024) (Cited on page 3, 4, 7, 8, 9, 10, 11, 12, 14, 21, 32.)

[8] 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 8, 10.)

[9] 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, Berlin, Heidelberg (May / Jun 2006) (Cited on page 10, 11, 29, 38, 39, 50, 51, 53, 55.)

[10] Bienstock, A., Patel, S., Seo, J.Y., Yeo, K.: Near-optimal oblivious key-value stores for efficient PSI, PSU and volume-hiding multi-maps. In: Calandrino, J.A., Troncoso, C. (eds.) USENIX Security 2023. pp. 301–318. USENIX Association (Aug 2023) (Cited on page 3, 4, 5, 13, 42, 43, 44, 45.)

[11] Boyle, E., Couteau, G., Gilboa, N., Ishai, Y.: Compressing vector OLE. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 896–912. ACM Press (Oct 2018) (Cited on page 32, 34.)

[12] Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators from ring-LPN. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 387–416. Springer, Cham (Aug 2020) (Cited on page 32.)

[13] de Castro, L., Juvekar, C., Vaikuntanathan, V.: Fast vector oblivious linear evaluation from ring learning with errors. Cryptology ePrint Archive, Report 2020/685 (2020) (Cited on page 32, 34.)

[14] 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, Cham (Aug 2020) (Cited on page 3, 4, 5, 8, 9, 13.)

[15] Chou, T., Orlandi, C.: The simplest protocol for oblivious transfer. In: International Conference on Cryptology and Information Security in Latin America. pp. 40–58. Springer (2015) (Cited on page 21, 23, 24.)

[16] Falk, B.H., Noble, D., Ostrovsky, R.: Private set intersection with linear communication from general assumptions. Cryptology ePrint Archive, Report 2018/238 (2018) (Cited on page 3, 8.)

[17] 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, Berlin, Heidelberg (Feb 2005) (Cited on page 3, 13.)

[18] 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, Berlin, Heidelberg (May 2004) (Cited on page 3, 9.)

[19] Garimella, G., Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: Oblivious key-value stores and amplification for private set intersection. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part II. LNCS, vol. 12826, pp. 395–425. Springer, Cham, Virtual Event (Aug 2021) (Cited on page 3, 4, 5, 9, 13, 32, 42, 43, 44, 45.)

[20] Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. Journal of the ACM 33(4), 792–807 (Oct 1986) (Cited on page 7, 13.)

[21] Google: Better password protections in chrome - how it works. https://security.googleblog.com/2019/12/better-password-protections-in-chrome.html (Dec 2019) (Cited on page 3.)

[22] Haitner, I., Segev, G.: Tighter concrete security for the simplest OT. CiC 2(1), 11 (2025) (Cited on page 24.)

[23] 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, Berlin, Heidelberg (Mar 2008) (Cited on page 3, 4, 7, 8.)

[24] Hetz, L., Schneider, T., Weinert, C.: Scaling mobile private contact discovery to billions of users. In: Tsudik, G., Conti, M., Liang, K., Smaragdakis, G. (eds.) ESORICS 2023, Part I. LNCS, vol. 14344, pp. 455–476. Springer, Cham (Sep 2023) (Cited on page 3.)

[25] Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the American statistical association 58(301), 13–30 (1963) (Cited on page 25.)

[26] Huberman, B.A., Franklin, M., Hogg, T.: Enhancing privacy and trust in electronic communities. In: Proceedings of the 1st ACM Conference on Electronic Commerce. p. 78–86. EC '99, Association for Computing Machinery, New York, NY, USA (1999) (Cited on page 3.)

[27] Hunt, T., Hunt, C., Siguroarson, S.: Have I been pwned? https://haveibeenpwned.com/ (Cited on page 3.)

[28] 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) (Cited on page 3.)

[29] 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) (Cited on page 3.)

[30] Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Berlin, Heidelberg (Aug 2003) (Cited on page 3, 7, 8, 21, 22.)

[31] 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, Berlin, Heidelberg (Dec 2014) (Cited on page 4.)

[32] 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, Berlin, Heidelberg (Mar 2009) (Cited on page 3.)

[33] Kales, D., Rechberger, C., Schneider, T., Senker, M., Weinert, C.: Mobile private contact discovery at scale. In: Heninger, N., Traynor, P. (eds.) USENIX Security 2019. pp. 1447–1464. USENIX Association (Aug 2019) (Cited on page 3.)

[34] Kirsch, A., Mitzenmacher, M., Wieder, U.: More robust hashing: Cuckoo hashing with a stash. SIAM Journal on Computing 39(4), 1543–1561 (2010) (Cited on page 5, 8.)

[35] 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 3, 4, 5, 6, 7, 8, 13, 16, 21, 22.)

[36] 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 3.)

[37] Marlinspike, M.: The difficulty of private contact discovery. https://signal.org/blog/contact-discovery/ (Jan 2014) (Cited on page 3.)

[38] Meadows, C.: A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In: 1986 IEEE Symposium on Security and Privacy. pp. 134–134 (1986) (Cited on page 3.)

[39] Micali, S., Pass, R., Rosen, A.: Input-indistinguishable computation. In: 47th FOCS. pp. 367–378. IEEE Computer Society Press (Oct 2006) (Cited on page 3.)

[40] Microsoft: Password monitor: Safeguarding passwords in microsoft edge. https://www.microsoft.com/en-us/research/blog/password-monitor-safeguarding-passwords-in-microsoft-edge (Jan 2021) (Cited on page 3.)

[41] Nagaraja, S., Mittal, P., Hong, C.Y., Caesar, M., Borisov, N.: BotGrep: Finding P2P bots with structured graph analysis. In: USENIX Security 2010. pp. 95–110. USENIX Association (Aug 2010) (Cited on page 3.)

[42] Naor, M., Pinkas, B.: Efficient oblivious transfer protocols. In: Kosaraju, S.R. (ed.) 12th SODA. pp. 448–457. ACM-SIAM (Jan 2001) (Cited on page 21, 23.)

[43] 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 3, 13.)

[44] 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 3.)

[45] Noble, D.: An intimate analysis of cuckoo hashing with a stash. Cryptology ePrint Archive, Report 2021/447 (2021) (Cited on page 19.)

[46] Orrù, M., Orsini, E., Scholl, P.: Actively secure 1-out-of-N OT extension with application to private set intersection. In: Handschuh, H. (ed.) CT-RSA 2017. LNCS, vol. 10159, pp. 381–396. Springer, Cham (Feb 2017) (Cited on page 3.)

[47] Pagh, R., Rodler, F.F.: Cuckoo hashing. Journal of Algorithms 51(2), 122–144 (2004) (Cited on page 5, 8.)

[48] 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, Cham (Aug 2019) (Cited on page 3, 4, 5, 9, 13.)

[49] Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: PSI from PaXoS: Fast, malicious private set intersection. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part II. LNCS, vol. 12106, pp. 739–767. Springer, Cham (May 2020) (Cited on page 3, 4, 5, 8, 9, 13, 43.)

[50] 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 3, 4, 5, 7, 8.)

[51] Pinkas, B., Schneider, T., Zohner, M.: Faster private set intersection based on OT extension. In: Fu, K., Jung, J. (eds.) USENIX Security 2014. pp. 797–812. USENIX Association (Aug 2014) (Cited on page 3, 4, 5, 8.)

[52] Pinkas, B., Schneider, T., Zohner, M.: Scalable private set intersection based on ot extension. ACM Transactions on Privacy and Security (TOPS) 21(2), 1–35 (2018) (Cited on page 5, 19.)

[53] Raghuraman, S., Rindal, P.: Blazing fast PSI from improved OKVS and subfield VOLE. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 2505–2517. ACM Press (Nov 2022) (Cited on page 3, 4, 5, 13, 32, 42, 43, 44, 45.)

[54] Rindal, P., Rosulek, M.: Malicious-secure private set intersection via dual execution. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 1229–1242. ACM Press (Oct / Nov 2017) (Cited on page 3.)

[55] Rindal, P., Schoppmann, P.: VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part II. LNCS, vol. 12697, pp. 901–930. Springer, Cham (Oct 2021) (Cited on page 3, 4, 5, 9, 13, 32, 42, 43.)

[56] Rosulek, M., Trieu, N.: Compact and malicious private set intersection for small sets. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 1166–1181. ACM Press (Nov 2021) (Cited on page 9.)

[57] Schoppmann, P., Gascón, A., Reichert, L., Raykova, M.: Distributed vector-OLE: Improved constructions and implementation. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 1055–1072. ACM Press (Nov 2019) (Cited on page 32, 34.)

[58] Wang, X., Lu, Z., Liang, B., Meng, S.: Unbalanced psi from client-independent relaxed oblivious prf. Proc. Priv. Enhancing Technol. 2025, 475–493 (2025) (Cited on page 7.)

[59] Weng, C., Yang, K., Katz, J., Wang, X.: Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In: 2021 IEEE Symposium on Security and Privacy. pp. 1074–1091. IEEE Computer Society Press (May 2021) (Cited on page 32, 34.)

[60] Yang, K., Weng, C., Lan, X., Zhang, J., Wang, X.: Ferret: Fast extension for correlated OT with small communication. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020. pp. 1607–1626. ACM Press (Nov 2020) (Cited on page 32, 34.)

[61] Yeo, K.: Cuckoo hashing in cryptography: Optimal parameters, robustness and applications. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part IV. LNCS, vol. 14084, pp. 197–230. Springer, Cham (Aug 2023) (Cited on page 5, 19.)

[62] Zhang, C., Chen, Y., Liu, W., Zhang, M., Lin, D.: Linear private set union from multi-query reverse private membership test. In: Calandrino, J.A., Troncoso, C. (eds.) USENIX Security 2023. pp. 337–354. USENIX Association (Aug 2023) (Cited on page 42.)

A Proof of Proposition 4.1

Proof of Proposition 4.1: Adversary $\mathrm{A}_{\mathrm{jit}}$ runs $c' \leftarrow \mathrm{A}_{\mathrm{prf}}[\mathrm{TR}, \mathrm{CH}, \mathrm{RO}]$ and returns c' as its output. Here, TR, CH are oracles available to $\mathrm{A}_{\mathrm{jit}}$ which it uses to replace oracles GenKey and Fn, respectively, that $\mathrm{A}_{\mathrm{prf}}$ expects. $\mathrm{A}_{\mathrm{jit}}$ also provides its RO to $\mathrm{A}_{\mathrm{prf}}$. The TR oracle initializes the keys and sets the table entries of T same as in GenKey. The code of CH is exactly the same as Fn. Thus, whenever $\mathrm{A}_{\mathrm{prf}}$ wins $\mathrm{A}_{\mathrm{jit}}$ wins and we have the claimed result in Eq. (1). ∎

Game G

INITIALIZE:
1 $\mathsf{H} \leftarrow_{\$} \mathsf{OS}$

RUN$((S_1, s_2), (S_2, s_1))$: // $s_1 = |S_1|$ and $s_2 = |S_2|$ by functionality definition
2 $\mathbf{x} \leftarrow \mathsf{S2V}(S_1)$
3 $((y_1, \ldots, y_{s_1}), (K_1, \ldots, K_n)) \leftarrow \mathsf{F}_{\mathsf{f}}^{\text{jit-oprf}}[\mathsf{H}]((\mathbf{x}[1], \ldots, \mathbf{x}[s_1]), s_1)$ // $n = \mathsf{f.nk}(\mathsf{f.cp}, s_1)$
4 $Z \leftarrow \{\mathsf{f.Ev}[\mathsf{H}](K, x) \mid K \in \{K_1, \ldots, K_n\}$ and $x \in S_2\}$
5 $I \leftarrow \{\mathbf{x}[i] \mid i \in [s_1], y_i \in Z\}$
6 $(Y_1, Y_2) \leftarrow \mathsf{F}(S_1, S_2)$
7 If $(Y_1 \neq I)$ or $(Y_2 \neq \varepsilon)$ then $\mathsf{win} \leftarrow 1$
8 Return win

RO(X):
9 Return $\mathsf{H}(X)$

FINALIZE:
10 Return win

Figure 28: Game for proof of Theorem 5.1 part (1).

B Proof of Theorem 5.1

B.1 Proof for Theorem 5.1 Part (1)

Proof of Theorem 5.1, Part (1): We will use the game, G, shown in Figure 28. The game G is same as the game $\mathbf{G}^{\text{corr}}_{\mathsf{F},\Pi}$ with the protocol steps expanded in RUN with a slight difference arising due to line 3. In G, instead of the using the JIT-OPRF protocol Π^{jit}, the functionality $\mathsf{F}_{\mathsf{f}}^{\text{jit-oprf}}$ is used to compute the JIT-OPRF outputs. Since Π^{jit} is perfectly correct, this change has no effect. So we get, $\Pr[\mathrm{G}(\mathrm{A}_{\text{psi}})] = \Pr[\mathbf{G}^{\text{corr}}_{\mathsf{F},\Pi}(\mathrm{A}_{\text{psi}})]$.

Now, to bound $\Pr[\mathrm{G}(\mathrm{A}_{\text{psi}})]$, we construct adversary A_{prf} playing the game $\mathbf{G}^{\text{jit-prf}}_{\mathsf{f}}$ using adversary A_{psi}. Adversary A_{prf} initializes $i \leftarrow 0$, $\mathsf{win} \leftarrow 0$ and runs $\mathrm{A}_{\text{psi}}[\mathrm{RO}, \textsc{Run}]$. Finally, A_{prf} returns a bit $b \leftarrow [[\mathsf{win} = 1]]$ as its output. Here, RO is the random oracle available to A_{prf} that it supplies to A_{psi}. This is fine as $\mathsf{f}, \Pi^{\text{jit}}$ and Π use the same random oracle. However, RUN is a subroutine that A_{prf} defines to replace the oracle of the same name that A_{psi} expects. In defining RUN, A_{prf} uses its own oracles GENKEY and FN, as follows:

RUN$((S_1, s_2), (S_2, s_1))$: // $s_1 = |S_1|$ and $s_2 = |S_2|$
1 $\mathbf{x} \leftarrow \mathsf{S2V}(S_1)$
2 GENKEY$(\mathbf{x}[1], \ldots, \mathbf{x}[s_1])$; $i \leftarrow i + 1$
3 For $j = 1, \ldots, s_1$ do
4 $\quad y_j \leftarrow \textsc{Fn}(i, \mathsf{f.nk}(\mathsf{f.cp}, j), \mathbf{x}[j])$
5 $Z \leftarrow \{\textsc{Fn}(i, j, x) \mid j \in [1..\mathsf{f.nk}(\mathsf{f.cp}, s_1)], x \in S_2\}$
6 $I \leftarrow \{\mathbf{x}[j] \mid j \in [1..s_1], y_j \in Z\}$
7 $(Y_1, Y_2) \leftarrow \mathsf{F}(S_1, S_2)$
8 If $(Y_1 \neq I)$ or $(Y_2 \neq \varepsilon)$ then $\mathsf{win} \leftarrow 1$
9 Return win

As mentioned in the correctness definition, we assume that A_{psi} queries RUN oracle with valid inputs which gives $s_1 = |S_1|$ and $s_2 = |S_2|$. Here, A_{prf} uses its GENKEY oracle to initialize a new key tuple based on the inputs $(\mathbf{x}[1], \ldots, \mathbf{x}[s_1])$. It constructs the output values $y_1, \ldots, y_{s_1}$ and the set Z using its FN oracle. The rest of the operations are similar to the RUN in G with line 3 removed. Finally, A_{prf} returns a bit $b \leftarrow [[\mathsf{win} = 1]]$ as its output.

Let b' be the challenge bit initialized in $\mathbf{G}^{\text{jit-prf}}_{\mathsf{f}}$. When $b' = 1$, the output of FN oracle is computed using $\mathsf{f.Ev}$. So, from line 4 in the subroutine above, $y_j = \mathsf{f.Ev}[\text{RO}](K_{\mathsf{f.nk}(\mathsf{f.cp},j)}, \mathbf{x}[j])$ for all $j \in [1..s_1]$ where the keys $(K_1, \ldots, K_n)$ were initialized through line 2 using the values in $\mathbf{x}$. This means the output values $y_1, \ldots, y_{s_1}$ and keys $(K_1, \ldots, K_n)$ are constructed as needed, that is, identical to functionality $\mathsf{F}^{\text{jit-oprf}}_{\mathsf{f}}$ outputs, which is same as line 3 in G. The above makes A_{prf} simulate G.RUN oracle for A_{psi} correctly. Therefore, we get $\Pr[\, b = 1 \mid b' = 1 \,] = \Pr[\text{G}(\text{A}_{\text{psi}})]$.

In the other case, when $b' = 0$, the output of FN oracle is a random value from Rng. We know that A_{psi} sets the win flag only if $Y_1 \neq I$ (since $Y_2 = \varepsilon$ always). In a RUN oracle query, this only happens if the output of $\text{FN}(i, j, x)$ for some $x \in S_2 \setminus (S_1 \cap S_2)$ is the same as some value in $\{y_1, \ldots, y_{s_1}\}$. These values are uniformly picked from Rng. We proceed to upper-bound the probability of this occurring.

Let $q = \text{Q}^{\text{RUN}}(\text{A}_{\text{psi}})$. For the i-th $\text{RUN}((S_1, s_2), (S_2, s_1))$ call by A_{psi}, let $s_{i,j} = s_j$ and $n_i = \mathsf{f.nk}(\mathsf{f.cp}, s_{i,1})$. For each element $x \in S_2$, the probability that a collision as described above happens is at most $s_{i,1}/|\mathsf{Rng}|$ in each $\text{FN}(i, j, x)$ call. Since this call happens n_i times (number of keys) the bound becomes $n_i \cdot s_{i,1}/|\mathsf{Rng}|$. Using union bound across all the elements in S_2 across all queries we get,

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

Thus, we get

$$\mathbf{Adv}^{\text{jit-prf}}_{\mathsf{f}}(\text{A}_{\text{prf}}) \geq \Pr[\text{G}(\text{A}_{\text{psi}})] - \sum_{i=1}^{q} \frac{n_i s_{i,1} s_{i,2}}{|\mathsf{Rng}|} \ .$$

Combining this with the results above we get

$$\Pr[\mathbf{G}^{\text{corr}}_{\mathsf{F},\Pi}(\text{A}_{\text{psi}})] \leq \mathbf{Adv}^{\text{jit-prf}}_{\mathsf{f}}(\text{A}_{\text{prf}}) + \sum_{i=1}^{q} \frac{n_i s_{i,1} s_{i,2}}{|\mathsf{Rng}|} \ ,$$

which is Eq. (2).

For resource usage, we note that RO oracle is only called by A_{prf} is A_{psi} makes a random oracle call and so $\text{Q}^{\text{RO}}(\text{A}_{\text{prf}}) = \text{Q}^{\text{RO}}(\text{A}_{\text{psi}})$. In the i-th $\text{RUN}((S_1, s_2), (S_2, s_1))$ query by A_{psi}, let $s_{i,j} = s_j$ and $n_i = \mathsf{f.nk}(\mathsf{f.cp}, s_{i,1})$. Then from the subroutine shown above, we get $\text{Q}^{\text{GENKEY}}(\text{A}_{\text{prf}}) = \text{Q}^{\text{RUN}}(\text{A}_{\text{psi}})$ and $\text{Q}^{\text{FN}}(\text{A}_{\text{prf}}) = \sum_{i=1}^{q} s_{i,1} + n_i \cdot s_{i,2}$ where $q = \text{Q}^{\text{RUN}}(\text{A}_{\text{psi}})$. ∎

B.2 Proof for Theorem 5.1 Part (2)

Proof of Theorem 5.1, Part (2): To show client (party 1) security we construct adversary A_{jit} playing the game $\mathbf{G}^{\text{jit-oprf-1}}_{\mathsf{f},\Pi^{\text{jit}}}$ using adversary A_{psi} playing the game $\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,1}$. Adversary A_{jit} runs $b \leftarrow\!\!\$\ \text{A}_{\text{psi}}[\text{RO}, \text{RUN}]$ and returns b. The random oracle used in Π is the same as the one used in f and Π^{jit}. Thus, A_{jit} supplies to A_{psi} the random oracle RO available to it. Here, RUN is a subroutine which replaces the oracle of the same name that A_{psi} expects. A_{jit} constructs it using its CH oracle as follows:

$\underline{\text{RUN}((S_{1,0}, s_2), (S_{1,1}, s_2), (S_2, s_1))}$: // $|S_{1,0}| = |S_{1,1}| = s_1$

1 $\mathbf{x}_0 \leftarrow \mathsf{S2V}(S_{1,0})$; $\mathbf{x}_1 \leftarrow \mathsf{S2V}(S_{1,1})$

2 $((K_1, \ldots, K_n), \tau, \omega) \leftarrow \text{CH}((\mathbf{x}_0[1], \ldots, \mathbf{x}_0[s_1]), (\mathbf{x}_1[1], \ldots, \mathbf{x}_1[s_1]))$

3 $Z \leftarrow \{\mathsf{f.Ev}[\mathrm{RO}](K, x) \mid K \in \{K_1, \ldots, K_n\}, x \in S_2\}$
4 Return $(\tau \| Z, \omega)$

Since we assume that adversaries provide valid inputs in their queries, here, $|S_{1,0}| = |S_{1,1}| = s_1$. Adversary $\mathrm{A}_{\mathrm{jit}}$ creates the transcript of executing Π for either $((S_{1,0}, s_2), (S_2, s_1))$ or $((S_{1,1}, s_2), (S_2, s_1))$. For this, it constructs the vectors $\mathbf{x}_0$ and $\mathbf{x}_1$ from $S_{1,0}$ and $S_{1,1}$ respectively. It gets the transcript of executing Π^{jit} using its CH oracle query on these inputs, which also outputs the keys $(K_1, \ldots, K_n)$. $\mathrm{A}_{\mathrm{jit}}$ completes the transcript by constructing the set Z using these keys and S_2. Clearly the transcript returned to $\mathrm{A}_{\mathrm{psi}}$ is correct and the challenge bit in $\mathbf{G}^{\text{jit-oprf-1}}_{\mathsf{f},\Pi^{\mathrm{jit}}}$ selects the input set used ($S_{1,0}$ or $S_{1,1}$).

As the challenge bit for $\mathrm{A}_{\mathrm{psi}}$ is determined by the challenge bit for $\mathrm{A}_{\mathrm{jit}}$, if $\mathrm{A}_{\mathrm{psi}}$ wins then $\mathrm{A}_{\mathrm{jit}}$ wins too. So we get the claimed result $\mathbf{Adv}^{\mathrm{ini}}_{\mathsf{F},\Pi,1}(\mathrm{A}_{\mathrm{psi}}) \leq \mathbf{Adv}^{\text{jit-oprf-1}}_{\mathsf{f},\Pi^{\mathrm{jit}}}(\mathrm{A}_{\mathrm{jit}})$ which is Eq. (3).

Note that if there was a correctness error associated with Π^{jit} then it would show up as a term in the above result. This is because an adversary can attack the scheme by using any input that leads to the correctness error as one side of the inputs in CH oracle. So the reduction would need to incorporate this, which would introduce the correctness error term. However, Π^{jit} is perfectly correct, so this is not needed in the reduction above.

For resource usage, let $q = \mathrm{Q}^{\mathrm{Run}}(\mathrm{A}_{\mathrm{psi}})$. See that $\mathrm{A}_{\mathrm{jit}}$ makes one CH oracle call for every Run query made by $\mathrm{A}_{\mathrm{psi}}$ and so $\mathrm{Q}^{\mathrm{CH}}(\mathrm{A}_{\mathrm{jit}}) = q$. Also, $\mathrm{A}_{\mathrm{jit}}$ makes a Run oracle query for every $\mathrm{A}_{\mathrm{psi}}$ random oracle query. We get $\mathrm{Q}^{\mathrm{RO}}(\mathrm{A}_{\mathrm{jit}}) = \mathrm{Q}^{\mathrm{RO}}(\mathrm{A}_{\mathrm{psi}})$. Note that as per our convention the random oracle calls made in line 3 of the subroutine Run that $\mathrm{A}_{\mathrm{jit}}$ creates would be counted in the random oracle calls of $\mathrm{A}_{\mathrm{psi}}$ as these are also made in the game $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F},\Pi,1}$. ∎

B.3 Proof for Theorem 5.1 Part (3)

Proof of Theorem 5.1, Part (3): To show server (party 2) security we construct adversary $\mathrm{A}_{\mathrm{jit}}$ playing the game $\mathbf{G}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi^{\mathrm{jit}}}$ using $\mathrm{A}_{\mathrm{psi}}$ playing the game $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F},\Pi,2}$. Adversary $\mathrm{A}_{\mathrm{jit}}$ starts by uniformly selecting a bit c and initializing $i \leftarrow 0$. It then runs $c' \leftarrow\!\!\$\ \mathrm{A}_{\mathrm{psi}}[\mathrm{RO}, \mathrm{Run}]$ and returns $[[c' = c]]$. $\mathrm{A}_{\mathrm{jit}}$ supplies its own RO to $\mathrm{A}_{\mathrm{psi}}$ as Π and $\mathsf{f}, \Pi^{\mathrm{jit}}$ use the same random oracle. Here, Run is a subroutine which replaces the oracle of the same name that $\mathrm{A}_{\mathrm{psi}}$ expects. $\mathrm{A}_{\mathrm{jit}}$ constructs it using its TR and CH oracles as follows:

$\underline{\mathrm{Run}((S_{2,0}, s_1), (S_{2,1}, s_1), (S_1, s_2))}$: // $|S_{2,0}| = |S_{2,1}| = s_2$
1 If $S_1 \cap S_{2,0} \neq S_1 \cap S_{2,1}$ then return $\perp$
2 $\mathbf{x} \leftarrow \mathsf{S2V}(S_1)$; $i \leftarrow i + 1$
3 $((y_1, \ldots, y_{s_1}), \tau, \omega) \leftarrow \mathrm{TR}((\mathbf{x}[1], \ldots, \mathbf{x}[s_1]))$
4 For $j = 1, \ldots, s_1$ do
5 If $\mathbf{x}[j] \in S_{2,c}$ then $Z \leftarrow Z \cup \{y_j\}$
6 For $x \in S_{2,c} \setminus S_1 \cap S_{2,c}$ do
7 $Z \leftarrow Z \cup \{\mathrm{CH}(i, j, x) \mid j \in [1..\mathsf{f.nk}(\mathsf{f.cp}, s_1)]\}$
8 Return $(\tau \| Z, \omega)$

Since we assume that adversaries provide valid inputs in their queries, here, $|S_{2,0}| = |S_{2,1}| = s_2$. In the subroutine above, $\mathrm{A}_{\mathrm{jit}}$ first makes the required check done in $\mathbf{G}^{\mathrm{ini}}_{\mathsf{F},\Pi,2}$.Run. To construct the transcript it uses the sets S_1 and $S_{2,c}$. Using S_1 and its TR oracle, adversary $\mathrm{A}_{\mathrm{jit}}$ gets the transcript (along with the random coins for party 1) for executing Π^{jit} as well as the outputs $(y_1, \ldots, y_{s_1})$. This also creates the keys, corresponding to values in $\mathbf{x}$, for the game to use. Consequently, $\mathrm{A}_{\mathrm{jit}}$

can now create the set Z using values in $(y_1, \ldots, y_{s_1})$ for elements in $S_1 \cap S_{2,c}$ and its CH oracle for rest of the elements in $S_{2,c}$. This completes the transcript for $\mathrm{A}_{\mathrm{psi}}$.

Let b' be the challenge bit initialized in game $\mathbf{G}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi^{\mathrm{jit}}}$. When $b' = 1$, the output of CH is computed using $\mathsf{f.Ev}$ using the appropriate key specified by indices i, j. Therefore, $\mathrm{A}_{\mathrm{jit}}$ correctly simulates the RUN oracle for $\mathrm{A}_{\mathrm{psi}}$. We get $\Pr[\, c' = c \mid b' = 1 \,] = \Pr[\mathbf{G}^{\mathrm{ini}}_{\mathsf{F},\Pi,2}(\mathrm{A}_{\mathrm{psi}})]$.

In the other case when $b' = 0$, the output of CH oracle is uniformly selected from Rng. Note that the output from CH oracle call would be identically distributed for any value in $S_{2,1} \cup S_{2,2} \setminus S_1 \cap S_{2,c}$. As a result, the values in Z obtained in line 7, in the subroutine above, do not provide any information about c. One may think that an adversary could get some information about c by repeating values used for honest party input sets in prior RUN calls. However, this does not work since every new RUN call increments i which is used in the CH call, which means newly initialized keys (through line 3) are used in each RUN call. As $S_1 \cap S_{2,0} = S_1 \cap S_{2,1}$ for a query resulting in a non-$\perp$ response, the operations before line 6 in the subroutine above are always independent of the challenge bit. Thus, adversary $\mathrm{A}_{\mathrm{psi}}$ can only guess the challenge bit in this case. We get $\Pr[\, c' = c \mid b' = 0 \,] = 1/2$. Combining with the above we get

$$\mathbf{Adv}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi^{\mathrm{jit}}}(\mathrm{A}_{\mathrm{jit}}) = \Pr[\mathbf{G}^{\mathrm{ini}}_{\mathsf{F},\Pi,2}(\mathrm{A}_{\mathrm{psi}})] - \frac{1}{2} \,,$$

which yields Eq. (4).

For resource usage, let $q = \mathrm{Q}^{\mathrm{RUN}}(\mathrm{A}_{\mathrm{psi}})$. See that $\mathrm{A}_{\mathrm{jit}}$ makes one TR oracle call for every RUN query made by $\mathrm{A}_{\mathrm{psi}}$ and so $\mathrm{Q}^{\mathrm{TR}}(\mathrm{A}_{\mathrm{jit}}) = q$. Let $s_{i,j}$ be the size of party j set and $n_i = \mathsf{f.nk}(\mathsf{f.cp}, s_{i,1})$ in i-th RUN oracle query. Then, $\mathrm{A}_{\mathrm{jit}}$ makes $n_i \cdot s_{i,2}$ CH oracle calls in i-th RUN query from $\mathrm{A}_{\mathrm{psi}}$. We get $\mathrm{Q}^{\mathrm{CH}}(\mathrm{A}_{\mathrm{jit}}) = \sum_{i=1}^{q} n_i \cdot s_{i,2} \leq n \cdot s_2$ where s_2 and n are upper bounds on the sum of the sizes of the sets of party 2 and number of keys given to party 2, respectively, across all RUN queries. Since $\mathrm{A}_{\mathrm{jit}}$ supplies its RO oracle to $\mathrm{A}_{\mathrm{psi}}$, we get $\mathrm{Q}^{\mathrm{RO}}(\mathrm{A}_{\mathrm{jit}}) = \mathrm{Q}^{\mathrm{RO}}(\mathrm{A}_{\mathrm{psi}})$. ∎

C Proof of Theorem 5.2

C.1 Proof for Theorem 5.2 Part (1)

Proof of Theorem 5.2 Part (1): We will use the games, G_0 and G_1, shown in Figure 29. The game G_0 is same as the game $\mathbf{G}^{\mathrm{corr}}_{\mathsf{F},\Pi}$ with the protocol steps expanded in RUN with a slight difference arising due to line 10. Note that the random oracle RO allows querying G and H by supplying $a = \mathtt{g}$ and $a = \mathtt{h}$, respectively, where a is the first value in inputs to RO oracle. In G_0, instead of the using the JIT-OPRF protocol Π^{jit}, the functionality $\mathsf{F}^{\text{jit-oprf}}_{\mathsf{f}}$ is used to compute the JIT-OPRF outputs. Since Π^{jit} is perfectly correct, this change has no effect. So we get, $\Pr[\mathrm{G}_0(\mathrm{A}_{\mathrm{psi}})] = \Pr[\mathbf{G}^{\mathrm{corr}}_{\mathsf{F},\Pi}(\mathrm{A}_{\mathrm{psi}})]$. Note that input to RUN oracle which leads to error in bin allocation also leads to a win as this corresponds to party 1 setting its final decision in Π to 0 which, in $\mathbf{G}^{\mathrm{corr}}_{\mathsf{F},\Pi}$, sets the win flag.

Going from G_0 to G_1, the differences arise only if the bad flag is set (line 4). Instead of just setting the win flag as in G_0, in G_1 $\perp$ is returned. This means that win flag in G_1 can only be set in line 16 which checks if functionality outputs are different from outputs from Π. Since the games G_0 and G_1 are identical-until-bad, using Lemma 2.1 we get,

$$\Pr[\mathrm{G}_0(\mathrm{A}_{\mathrm{psi}})] - \Pr[\mathrm{G}_1(\mathrm{A}_{\mathrm{psi}})] \leq \Pr[\mathrm{G}_1(\mathrm{A}_{\mathrm{psi}}) \text{ sets } \mathsf{bad}] \,,$$

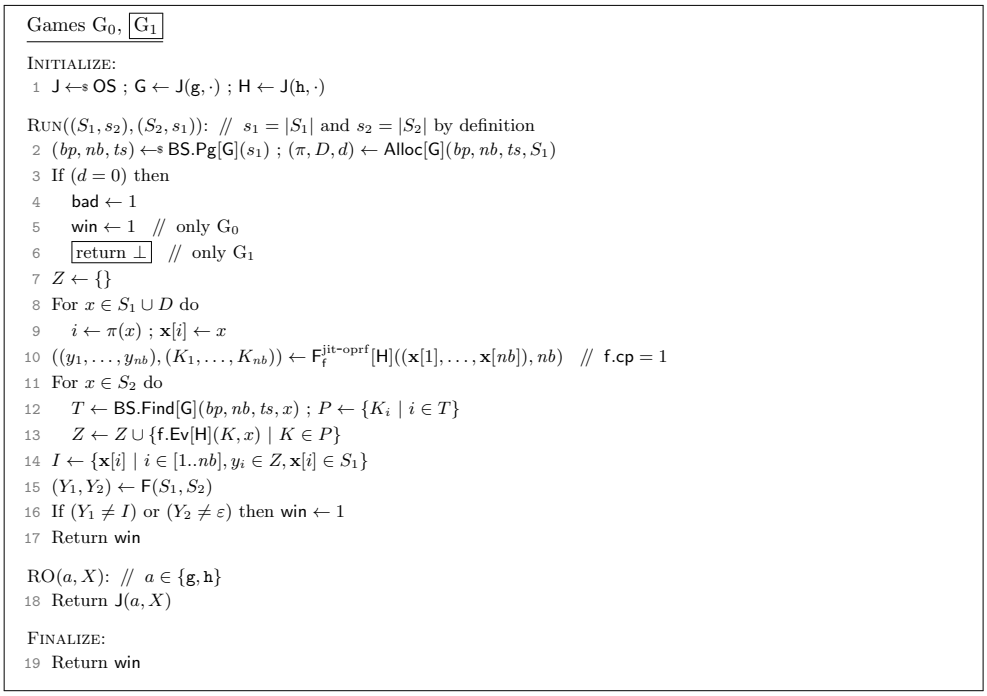


Figure 29: Games for proof of Theorem 5.2 part (1).

---

which gives

$$\Pr[\mathbf{G}^{\text{corr}}_{\mathsf{F},\Pi}(\mathrm{A}_{\text{psi}})] \leq \Pr[\mathrm{G}_1(\mathrm{A}_{\text{psi}})] + \Pr[\mathrm{G}_1(\mathrm{A}_{\text{psi}}) \text{ sets } \mathsf{bad}] \ .$$

Next, we construct adversary A_{cba} playing the game $\mathbf{G}^{\text{corr-ba}}_{\mathsf{BS}}$ using A_{psi}. A_{cba} runs $c' \leftarrow \mathrm{A}_{\text{psi}}[\mathrm{RO}, \textsc{Run}]$ and ignores c'. Here, $\textsc{ROSim}, \textsc{Run}$ are subroutines to replace the $\mathrm{RO}, \textsc{Run}$ oracles, respectively, that A_{psi} expects. A_{cba} creates $\textsc{Run}$ by copying the code of $\mathrm{G}_1.\textsc{Run}$ and replacing line 2 with the following code:

1. $(\mathsf{win}, \omega) \leftarrow \textsc{Alloc}(S_1)$ // S_1 from the query argument
2. $(bp, nb, ts) \leftarrow \mathsf{BS.Pg}[\mathrm{RO}](|S_1|; \omega)$
3. $(\pi, D, d) \leftarrow \mathsf{BS.Alloc}[\mathrm{RO}](bp, nb, ts, S_1)$

It also removes line 4 and replaces G in line 12 with its random oracle RO. The subroutine $\textsc{ROSim}$ is created as follows:

$\textsc{ROSim}(a, X)$:

```
1 If (a = g) then return RO(X)
2 Else
3   If T[a, X] = ⊥ then
4     T[a, X] ←$ Rng_h   // Rng_h is the range of functions in f.OS_2
5   return T[a, X]
```

The above makes A_{cba} correctly simulate $\mathrm{G}_1.\textsc{Run}$ as the code is exactly the same when expanded. If A_{psi} sets bad in G_1 then A_{cba} sets the win flag in the $\mathbf{G}^{\text{corr-ba}}_{\mathsf{BS}}$. So we get, $\Pr[\mathrm{G}_1(\mathrm{A}_{\text{psi}}) \text{ sets } \mathsf{bad}] = \Pr[\mathbf{G}^{\text{corr-ba}}_{\mathsf{BS}}(\mathrm{A}_{\text{cba}})] = \mathbf{Adv}^{\text{corr-ba}}_{\mathsf{BS}}(\mathrm{A}_{\text{cba}})$.

Now, to bound $\Pr[\mathrm{G}_1(\mathrm{A}_{\text{psi}})]$, we construct adversary A_{prf} playing the game $\mathbf{G}^{\text{jit-prf}}_{\mathsf{f}}$ using adversary A_{psi}. Adversary A_{prf} initializes $i \leftarrow 0$, $\mathsf{win} \leftarrow 0$, and runs $\mathrm{A}_{\text{psi}}[\textsc{ROSim}, \textsc{Run}]$. Finally, A_{prf} returns

a bit $b \leftarrow [[\mathsf{win} = 1]]$ as its output. Here, ROSIM, RUN are subroutines to replace the RO, RUN oracles, respectively, that $\mathrm{A}_{\mathrm{psi}}$ expects. In defining RUN, $\mathrm{A}_{\mathrm{prf}}$ uses its own oracles GENKEY and FN, as follows:

RUN$((S_1, s_2), (S_2, s_1))$: // $s_1 = |S_1|$ and $s_2 = |S_2|$

1 $(bp, nb, ts) \leftarrow\!\!\$\ \mathsf{BS.Pg}[\mathrm{ROSIM}(\mathsf{g}, \cdot)](s_1)$; $(\pi, D, d) \leftarrow \mathsf{BS.Alloc}[\mathrm{ROSIM}(\mathsf{g}, \cdot)](bp, nb, ts, S_1)$
2 If $(d = 0)$ then return $\perp$
3 $Z \leftarrow \{\}$
4 For $x \in S_1 \cup D$ do
5 $\quad i \leftarrow \pi(x)$; $\mathbf{x}[i] \leftarrow x$
6 GENKEY$(\mathbf{x}[1], \ldots, \mathbf{x}[nb])$; $i \leftarrow i + 1$
7 For $j = 1, \ldots, nb$ do
8 $\quad y_j \leftarrow \mathrm{FN}(i, j, \mathbf{x}[j])$
9 For $x \in S_2$ do
10 $\quad T \leftarrow \mathsf{BS.Find}[\mathrm{ROSIM}(\mathsf{g}, \cdot)](bp, nb, ts, x)$; $Z \leftarrow Z \cup \{\mathrm{FN}(i, j, x) \mid j \in T\}$
11 $I \leftarrow \{\mathbf{x}[j] \mid j \in [1..nb], y_j \in Z, \mathbf{x}[j] \in S_1\}$; $(Y_1, Y_2) \leftarrow \mathsf{F}(S_1, S_2)$
12 If $(Y_1 \neq I)$ or $(Y_2 \neq \varepsilon)$ then $\mathsf{win} \leftarrow 1$
13 Return win

The subroutine ROSIM is defined as follows:

ROSIM(a, X):

1 If $(a = \mathtt{h})$ then return $\mathrm{RO}(X)$
2 Else
3 $\quad$ If $\mathrm{T}[a, X] = \perp$ then
4 $\quad\quad \mathrm{T}[a, X] \leftarrow\!\!\$\ \mathsf{Rng}_{\mathsf{g}}$ // $\mathsf{Rng}_{\mathsf{g}}$ is the range of functions in $\mathsf{BS.OS}$
5 $\quad$ return $\mathrm{T}[a, X]$

As mentioned in the correctness definition, we assume that $\mathrm{A}_{\mathrm{psi}}$ queries the RUN oracle with valid inputs which gives $s_1 = |S_1|$ and $s_2 = |S_2|$ in the RUN subroutine. There, $\mathrm{A}_{\mathrm{prf}}$ follows the code in the RUN oracle of G_1. The difference is that it uses its GENKEY oracle to initialize a new key tuple based on the inputs $(\mathbf{x}[1], \ldots, \mathbf{x}[nb])$. It then constructs the output values $y_1, \ldots, y_{nb}$ and the set Z using its FN oracle. The rest of the operations are similar to the RUN oracle in G_1 with line 10 removed.

Let b' be the challenge bit initialized in $\mathbf{G}^{\text{jit-prf}}_{\mathsf{f}}$. When $b' = 1$, the output of the FN oracle is computed using $\mathsf{f.Ev}$. So, from line 8 in the subroutine above, $y_j = \mathsf{f.Ev}[\mathrm{RO}](K_j, \mathbf{x}[j])$ for all $j \in [1..nb]$ where the keys $(K_1, \ldots, K_{nb})$ were initialized through line 6 using the values in $\mathbf{x}$. This means the output values $y_1, \ldots, y_{nb}$ and keys $(K_1, \ldots, K_{nb})$ are constructed as needed, that is, same as functionality $\mathsf{F}^{\text{jit-oprf}}_{\mathsf{f}}$ outputs, which is same as line 10 in G_1. The above makes $\mathrm{A}_{\mathrm{prf}}$ simulate the G_1.RUN oracle for $\mathrm{A}_{\mathrm{psi}}$ correctly. Therefore, we get $\Pr[\, b = 1 \mid b' = 1 \,] = \Pr[\mathrm{G}_1(\mathrm{A}_{\mathrm{psi}})]$.

In the other case, when $b' = 0$, the output of the FN oracle is a random value from Rng. We know that $\mathrm{A}_{\mathrm{psi}}$ sets the win flag only if $Y_1 \neq I$ (since $Y_2 = \varepsilon$ always). In any RUN query response, this only happens if the output of $\mathrm{FN}(i, j, x)$ for some $x \in S_2 \setminus (S_1 \cap S_2)$ is the same as some value in a s_1 sized subset of $\{y_1, \ldots, y_{nb}\}$ (we can ignore the values corresponding to elements in D). These values are uniformly picked from Rng. We proceed to upper-bound the probability of this occurring.

Let $q = \mathrm{Q}^{\mathrm{RUN}}(\mathrm{A}_{\mathrm{psi}})$. For the i-th RUN$((S_1, s_2), (S_2, s_1))$ query by $\mathrm{A}_{\mathrm{psi}}$, let $s_{i,j} = s_j$ and $t_i = ts$. For each element $x \in S_2$, the probability that a collision as described above happens is at most

$s_{i,1}/|\mathsf{Rng}|$ (note this is not $nb/|\mathsf{Rng}|$ as y-values corresponding to elements in D can be ignored) for each $\textsc{Fn}(i,j,x)$ call. This call happens t_i times (number of keys) as this the size of set of bins returned from the BS.Find algorithm. Therefore, the bound becomes $t_i \cdot s_{i,1}/|\mathsf{Rng}|$. Using union bound across all the elements in S_2 across all queries we get,

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

Thus, we get

$$\mathbf{Adv}^{\text{jit-prf}}_{\mathsf{f}}(\mathrm{A}_{\text{prf}}) \geq \Pr[\mathrm{G}_1(\mathrm{A}_{\text{psi}})] - \sum_{i=1}^{q} \frac{t_i s_{i,1} s_{i,2}}{|\mathsf{Rng}|} \ .$$

Combining this with the results above we get

$$\Pr[\mathbf{G}^{\text{corr}}_{\mathsf{F},\Pi}(\mathrm{A}_{\text{psi}})] \leq \mathbf{Adv}^{\text{jit-prf}}_{\mathsf{f}}(\mathrm{A}_{\text{prf}}) + \sum_{i=1}^{q} \frac{t_i s_{i,1} s_{i,2}}{|\mathsf{Rng}|} + \mathbf{Adv}^{\text{corr-ba}}_{\mathsf{BS}}(\mathrm{A}_{\text{cba}}) \ ,$$

which is Eq. (5).

Let $q = \mathrm{Q}^{\textsc{Run}}(\mathrm{A}_{\text{psi}})$. Then for resource usage of A_{jit}, we note that $\mathrm{Q}^{\mathrm{RO}}(\mathrm{A}_{\text{prf}}) \leq \mathrm{Q}^{\mathrm{RO}}(\mathrm{A}_{\text{psi}})$ due to the subroutine $\textsc{ROSim}$. In the i-th $\textsc{Run}((S_1, s_2), (S_2, s_1))$ query by A_{psi}, let $s_{i,j} = s_j$ and $t_i = ts$, which is output from BS.Pg. Then from the subroutine shown above, we get $\mathrm{Q}^{\textsc{GenKey}}(\mathrm{A}_{\text{prf}}) = q$ and $\mathrm{Q}^{\textsc{Fn}}(\mathrm{A}_{\text{prf}}) = \sum_{i=1}^{q} s_{i,1} + t_i \cdot s_{i,2}$ where $q = \mathrm{Q}^{\textsc{Run}}(\mathrm{A}_{\text{psi}})$.

For resource usage of A_{cba}, we get $\mathrm{Q}^{\textsc{Alloc}}(\mathrm{A}_{\text{cba}}) = q$ due to the $\textsc{Run}$ subroutine defined by A_{cba} and $\mathrm{Q}^{\mathrm{RO}}(\mathrm{A}_{\text{psi}}) \leq \mathrm{Q}^{\mathrm{RO}}(\mathrm{A}_{\text{psi}})$ due to the $\textsc{ROSim}$ subroutine. ▌

C.2 Proof for Theorem 5.2 Part (2)

Proof of Theorem 5.2 Part (2): We will use the games G_0 and G_1 shown in Figure 30. The game G_0 is the same as $\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,1}$ with the protocol Π expanded. We get $\Pr[\mathrm{G}_0(\mathrm{A}_{\text{psi}})] = \Pr[\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,1}(\mathrm{A}_{\text{psi}})]$. Note that the random oracle RO allows querying G and H by supplying $a = \mathsf{g}$ and $a = \mathsf{h}$, respectively, where a is the first value in inputs to the RO oracle. The games G_0 and G_1 are identical-until-bad. Using Lemma 2.1 we get,

$$\Pr[\mathrm{G}_0(\mathrm{A}_{\text{psi}})] - \Pr[\mathrm{G}_1(\mathrm{A}_{\text{psi}})] \leq \Pr[\mathrm{G}_1(\mathrm{A}_{\text{psi}}) \text{ sets } \mathsf{bad}] \ ,$$

which gives

$$\Pr[\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,1}(\mathrm{A}_{\text{psi}})] \leq \Pr[\mathrm{G}_1(\mathrm{A}_{\text{psi}})] + \Pr[\mathrm{G}_1(\mathrm{A}_{\text{psi}}) \text{ sets } \mathsf{bad}] \ .$$

To bound $\Pr[\mathrm{G}_1(\mathrm{A}_{\text{psi}}) \text{ sets } \mathsf{bad}]$, we construct adversary A_{cba} playing game $\mathbf{G}^{\text{corr-ba}}_{\mathsf{BS}}$ using A_{psi}. A_{cba} initializes $c \leftarrow\!\!\$\ \{0,1\}$, runs $c' \leftarrow \mathrm{A}_{\text{psi}}[\textsc{ROSim}, \textsc{Run}]$ and ignores c'. Here, $\textsc{ROSim}, \textsc{Run}$ are subroutines to replace the $\mathrm{RO}, \textsc{Run}$ oracles, respectively, that A_{psi} expects. A_{cba} creates $\textsc{Run}$ by copying the code of $\mathrm{G}_1.\textsc{Run}$ and replacing lines 2 and 3 with the following code:

1. $(\mathsf{win}_0, \omega_0) \leftarrow \textsc{Alloc}(S_{1,0})$ // $S_{1,0}$ from the query argument
2. $(\mathsf{win}_1, \omega_1) \leftarrow \textsc{Alloc}(S_{1,1})$ // $S_{1,1}$ from the query argument
3. $(bp_0, nb_0, ts_0) \leftarrow \mathsf{BS.Pg}[\mathrm{RO}](|S_{1,0}|, \omega_0)$
4. $(bp_1, nb_1, ts_1) \leftarrow \mathsf{BS.Pg}[\mathrm{RO}](|S_{1,1}|, \omega_1)$
5. $(\pi_0, D_0, d_0) \leftarrow \mathsf{BS.Alloc}[\mathrm{RO}](bp_0, nb_0, ts_0, S_{1,0})$
6. $(\pi_1, D_1, d_1) \leftarrow \mathsf{BS.Alloc}[\mathrm{RO}](bp_1, nb_1, ts_1, S_{1,1})$

Games G_0, $\boxed{\mathrm{G}_1}$

INITIALIZE:
1 $c \leftarrow \{0,1\}$; $\mathsf{J} \leftarrow\!\!\$\ \mathsf{OS}$; $\mathsf{G} \leftarrow \mathsf{J}(\mathtt{g},\cdot)$; $\mathsf{H} \leftarrow \mathsf{J}(\mathtt{h},\cdot)$

RUN$((S_{1,0},s_2),(S_{1,1},s_2),(S_2,s_1))$: // $s_1 = |S_{1,0}| = |S_{1,1}|$ and $s_2 = |S_2|$
2 $(bp_0, nb_0, ts_0) \leftarrow\!\!\$\ \mathsf{BS.Pg}[\mathsf{G}](|S_{1,0}|)$; $(bp_1, nb_1, ts_1) \leftarrow\!\!\$\ \mathsf{BS.Pg}[\mathsf{G}](|S_{1,1}|)$
3 $(\pi_0, D_0, d_0) \leftarrow \mathsf{BS.Alloc}[\mathsf{G}](bp_0, nb_0, ts_0, S_{1,0})$; $(\pi_1, D_1, d_1) \leftarrow \mathsf{BS.Alloc}[\mathsf{G}](bp_1, nb_1, ts_1, S_{1,1})$
4 If $((d_0 = 0)$ or $(d_1 = 0))$ then
5 $\quad \mathsf{bad} \leftarrow 1$
6 $\quad$ If $(d_c = 0)$ then return $(\varepsilon, \varepsilon)$ // only G_0
7 $\quad \boxed{\text{return } \bot}$ // only G_1
8 $Z \leftarrow \{\}$; $\tau \leftarrow (bp_c, nb_c, ts_c)$
9 For $x \in S_{1,c} \cup D_c$ do
10 $\quad i \leftarrow \pi_c(x)$; $\mathbf{x}[i] \leftarrow x$
11 $\omega_1, \omega_2 \leftarrow\!\!\$\ \Omega$
12 $(\tau', st_1, st_2) \leftarrow \mathbf{Int}_{\Pi^{\mathrm{jit}}}[\mathsf{H}]((\mathbf{x}[1], \ldots, \mathbf{x}[nb_c]), nb_c; \omega_1, \omega_2)$
13 $(y_1, \ldots, y_{nb_c}) \leftarrow st_1.\mathrm{out}$; $(K_1, \ldots, K_{nb_c}) \leftarrow st_2.\mathrm{out}$ // $\mathsf{f.cp} = 1$
14 For $x \in S_2$ do
15 $\quad T \leftarrow \mathsf{BS.Find}[\mathsf{G}](bp_c, nb_c, ts_c, x)$; $P \leftarrow \{K_i \mid i \in T\}$
16 $\quad Z \leftarrow Z \cup \{\mathsf{f.Ev}[\mathsf{H}](K, x) \mid K \in P\}$
17 Return $(\tau \| \tau' \| Z, \omega_c)$

RO(a, X): // $a \in \{\mathtt{g}, \mathtt{h}\}$
18 Return $\mathsf{J}(a, X)$

FINALIZE(c'):
19 Return $[[c' = c]]$

Figure 30: Games for proof of Theorem 5.2 part (2).

It also removes line 5 and replaces G in line 15 with its random oracle RO. The subroutine ROSIM is created as follows:

ROSIM(a, X):
1 If $(a = \mathtt{g})$ then return RO(X)
2 Else
3 $\quad$ If $\mathrm{T}[a, X] = \bot$ then
4 $\quad\quad \mathrm{T}[a, X] \leftarrow\!\!\$\ \mathsf{Rng}_{\mathtt{h}}$ // $\mathsf{Rng}_{\mathtt{h}}$ is the range of functions in $\mathsf{f.OS}_2$
5 $\quad$ return $\mathrm{T}[a, X]$

The above makes $\mathrm{A}_{\mathrm{cba}}$ correctly simulate G_1.RUN as the code is exactly the same when expanded. If $\mathrm{A}_{\mathrm{psi}}$ sets bad in G_1 then $\mathrm{A}_{\mathrm{cba}}$ sets the win flag in the $\mathbf{G}^{\text{corr-ba}}_{\mathsf{BS}}$. So we get,

$$\Pr[\mathrm{G}_1(\mathrm{A}_{\mathrm{psi}}) \text{ sets } \mathsf{bad}] = \Pr[\mathbf{G}^{\text{corr-ba}}_{\mathsf{BS}}(\mathrm{A}_{\mathrm{cba}})] = \mathbf{Adv}^{\text{corr-ba}}_{\mathsf{BS}}(\mathrm{A}_{\mathrm{cba}}) \ .$$

In G_1, after bad is set, line 7 is used which, instead of following the protocol as in G_0 (line 6), returns $\bot$. Therefore, in order to get any information about the challenge bit in G_1, an adversary would need to query with sets $S_{1,0}$ and $S_{1,1}$ for which bin allocation succeeds. We upper-bound $\Pr[\mathrm{G}_1(\mathrm{A}_{\mathrm{psi}})]$ by constructing adversary $\mathrm{A}_{\mathrm{jit}}$ playing the game $\mathbf{G}^{\text{jit-oprf-1}}_{\mathsf{f},\Pi^{\mathrm{jit}}}$ using adversary $\mathrm{A}_{\mathrm{psi}}$. Adversary $\mathrm{A}_{\mathrm{jit}}$ runs $b \leftarrow\!\!\$\ \mathrm{A}_{\mathrm{psi}}[\mathrm{ROSIM}, \mathrm{RUN}]$ and returns b. Here, ROSIM, RUN are subroutines to replace the RO, RUN oracles, respectively, that $\mathrm{A}_{\mathrm{psi}}$ expects. $\mathrm{A}_{\mathrm{jit}}$ constructs RUN using its CH oracle as follows:

RUN$((S_{1,0},s_2),(S_{1,1},s_2),(S_2,s_1))$: // $s_1 = |S_{1,0}| = |S_{1,1}|$ and $s_2 = |S_2|$
1 $(bp, nb, ts) \leftarrow\!\!\$\ \mathsf{BS.Pg}[\mathrm{ROSIM}(\mathtt{g},\cdot)](|S_{1,0}|)$; $(\pi_0, D_0, d_0) \leftarrow \mathsf{BS.Alloc}[\mathrm{ROSIM}(\mathtt{g},\cdot)](bp, nb, ts, S_{1,0})$
2 $(\pi_1, D_1, d_1) \leftarrow \mathsf{BS.Alloc}[\mathrm{ROSIM}(\mathtt{g},\cdot)](bp, nb, ts, S_{1,1})$
3 If $((d_0 = 0)$ or $d_1 = 0)$ then return $\bot$
4 $Z \leftarrow \{\}$; $\tau \leftarrow (bp, nb, ts)$
5 For $x \in S_{1,0} \cup D_0$ do

$i \leftarrow \pi_0(x)$; $\mathbf{x}_0[i] \leftarrow x$
For $x \in S_{1,1} \cup D_1$ do
$i \leftarrow \pi_1(x)$; $\mathbf{x}_1[i] \leftarrow x$
$((K_1, \ldots, K_{nb}), \tau', \omega) \leftarrow \text{CH}((\mathbf{x}_0[1], \ldots, \mathbf{x}_0[nb]), (\mathbf{x}_1[1], \ldots, \mathbf{x}_1[nb]))$
For $x \in S_2$ do
$T \leftarrow \mathsf{BS.Find}[\text{ROSIM}(\mathsf{g}, \cdot)](bp, nb, ts, x)$; $P \leftarrow \{K_i \mid i \in T\}$
$Z \leftarrow Z \cup \{\mathsf{f.Ev}[\text{RO}](K, x) \mid K \in P\}$
Return $(\tau \| \tau' \| Z, \omega)$

The subroutine ROSIM is defined as follows:

ROSIM(a, X):
If $(a = \mathtt{h})$ then return RO(X)
Else
If T$[a, X] = \perp$ then
T$[a, X] \leftarrow\!\!\$\ \mathsf{Rng_g}$ // $\mathsf{Rng_g}$ is the range of functions in $\mathsf{BS.OS}$
return T$[a, X]$

Since we assume that adversaries provide valid inputs in their queries, $|S_{1,0}| = |S_{1,1}| = s_1$ in the RUN subroutine. There, adversary A_{jit} mostly follows the code in G_1.RUN. The first difference arises due to line 1 in the subroutine. Here, instead of running $\mathsf{BS.Pg}$ twice, as in G_1.RUN, it is just run once on $|S_{1,0}|$. This change has no effect since $|S_{1,0}| = |S_{1,1}|$ and so the two parameter generations in G_1.RUN are identical. Note that two parameter generations were needed in G_1.RUN to allow the reduction to the bin allocation correctness game that we showed earlier. The next difference in the subroutine is that it creates vectors $\mathbf{x}_0$ and $\mathbf{x}_1$ corresponding to the inputs $S_{1,0}$ and $S_{1,1}$ respectively. It then uses its CH oracle to get the keys, transcript and coins corresponding to executing protocol Π^{jit} on either the values in $\mathbf{x}_0$ or $\mathbf{x}_1$ based on the challenge bit in $\mathbf{G}^{\text{jit-oprf-1}}_{\mathsf{f},\Pi^{\text{jit}}}$. Therefore, if A_{psi} wins then A_{jit} wins too. So we get, $\Pr[\text{G}_1(\text{A}_{\text{psi}})] = \Pr[\mathbf{G}^{\text{jit-oprf-1}}_{\mathsf{f},\Pi^{\text{jit}}}(\text{A}_{\text{jit}})]$. Combining the results from above, we get

$$\Pr[\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,1}(\text{A}_{\text{psi}})] \leq \Pr[\mathbf{G}^{\text{jit-oprf-1}}_{\mathsf{f},\Pi^{\text{jit}}}(\text{A}_{\text{jit}})] + \mathbf{Adv}^{\text{corr-ba}}_{\mathsf{BS}}(\text{A}_{\text{cba}}) \ ,$$

which yields Eq. (6).

Let $q = \text{Q}^{\text{RUN}}(\text{A}_{\text{psi}})$. See that A_{jit} makes one CH oracle call for every RUN query made by A_{psi} and so $\text{Q}^{\text{CH}}(\text{A}_{\text{jit}}) = q$. Since A_{jit} defines ROSIM subroutine using its RO, we get $\text{Q}^{\text{RO}}(\text{A}_{\text{jit}}) \leq \text{Q}^{\text{RO}}(\text{A}_{\text{psi}})$.

For resource usage of A_{cba}, we get $\text{Q}^{\text{ALLOC}}(\text{A}_{\text{cba}}) = 2q$ due to the RUN subroutine that A_{cba} defines. Also, $\text{Q}^{\text{RO}}(\text{A}_{\text{psi}}) \leq \text{Q}^{\text{RO}}(\text{A}_{\text{psi}})$ due to the ROSIM subroutine. ▌

C.3 Proof for Theorem 5.2 Part (3)

Proof of Theorem 5.2 Part (3): To show server (party 2) security we construct adversary A_{jit} playing the game $\mathbf{G}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi^{\text{jit}}}$ using A_{psi} playing the game $\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,2}$. Adversary A_{jit} starts by uniformly selecting a bit c and initializing $i \leftarrow 0$. It then runs $c' \leftarrow\!\!\$\ \text{A}_{\text{psi}}[\text{ROSIM}, \text{RUN}]$ and returns $[[c' = c]]$. Here, ROSIM, RUN are subroutines to replace the RO, RUN oracles, respectively, that A_{psi} expects. A_{jit} constructs RUN using its TR and CH oracles as follows:

RUN$((S_{2,0}, s_1), (S_{2,1}, s_1), (S_1, s_2))$: // $s_2 = |S_{2,0}| = |S_{2,1}|$ and $s_1 = |S_1|$

1 If $S_1 \cap S_{2,0} \neq S_1 \cap S_{2,1}$ then return $\bot$
2 $(bp, nb, ts) \leftarrow\!\!\$\ \mathsf{BS.Pg}[\text{ROSIM}(\mathsf{g}, \cdot)](|S_1|)$; $(\pi, D, d) \leftarrow \mathsf{BS.Alloc}[\text{ROSIM}(\mathsf{g}, \cdot)](bp, nb, ts, S_1)$
3 If $(d = 0)$ then return $(\varepsilon, \varepsilon)$
4 For $x \in S_1 \cup D$ do
5 $\quad i \leftarrow \pi(x)$; $\mathbf{x}[i] \leftarrow x$
6 $i \leftarrow i + 1$; $Z \leftarrow \{\}$; $\tau \leftarrow (bp, nb)$
7 $((y_1, \ldots, y_{nb}), \tau', \omega) \leftarrow \text{TR}((\mathbf{x}[1], \ldots, \mathbf{x}[nb]))$
8 For $j = 1, \ldots, nb$ do
9 $\quad$ If $\mathbf{x}[j] \in S_{2,c}$ then $Z \leftarrow Z \cup \{y_j\}$
10 For $x \in S_{2,c} \setminus S_1 \cap S_{2,c}$ do
11 $\quad T \leftarrow \mathsf{BS.Find}[\text{ROSIM}(\mathsf{g}, \cdot)](bp, nb, ts, x)$; $Z \leftarrow Z \cup \{\text{CH}(i, j, x) \mid j \in T\}$
12 Return $(\tau \| \tau' \| Z, \omega)$

The subroutine ROSIM is defined as follows:

ROSIM(a, X):
1 If $(a = \mathtt{h})$ then return $\text{RO}(X)$
2 Else
3 $\quad$ If $\text{T}[a, X] = \bot$ then
4 $\quad\quad \text{T}[a, X] \leftarrow\!\!\$\ \mathsf{Rng_g}$ // $\mathsf{Rng_g}$ is the range of functions in $\mathsf{BS.OS}$
5 $\quad$ return $\text{T}[a, X]$

Note that the InI game for A_{psi} needs to provide its random oracle such that A_{psi} can query random oracles for BS and $\mathsf{f}, \Pi^{\text{jit}}$ separately. Here, we assume that it does so using the first value in the input, a.

Since we assume that adversaries provide valid inputs in their queries, $|S_{2,0}| = |S_{2,1}| = s_2$ in the RUN subroutine above. There, A_{jit} first makes the required check done in $\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,2}$.RUN. To construct the transcript it uses the sets S_1 and $S_{2,c}$. First it allocates the set S_1 using BS. If allocation fails, $(\varepsilon, \varepsilon)$ is returned similar to Π. Note that since allocation is only for the dishonest party set, S_1, inducing correctness error in BS would not provide any information about the challenge bit c. Adversary A_{jit} then constructs $\mathbf{x}$ same as Π and uses its TR oracle to get the transcript (along with the random coins for party 1) for executing Π^{jit} on $(\mathbf{x}[1], \ldots, \mathbf{x}[nb])$ as well as the outputs $(y_1, \ldots, y_{nb})$. This also creates the keys, corresponding to values in $\mathbf{x}$, for the game to use. Consequently, A_{jit} can now create the set Z using values in $(y_1, \ldots, y_{nb})$ for elements in $S_1 \cap S_{2,c}$ and its CH oracle for rest of the elements in $S_{2,c}$. This completes the transcript for A_{psi}.

Let b' be the challenge bit initialized in game $\mathbf{G}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi^{\text{jit}}}$. When $b' = 1$, the output of CH is computed using $\mathsf{f.Ev}$ using the appropriate key specified by indices i, j. Therefore, A_{jit} correctly simulates the RUN oracle for A_{psi}. We get $\Pr[\, c' = c \mid b' = 1 \,] = \Pr[\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,2}(\text{A}_{\text{psi}})]$.

In the other case when $b' = 0$, the output of CH is uniformly selected from Rng. Note that the output from CH oracle call would be identically distributed for any value in $S_{2,1} \cup S_{2,2} \setminus S_1 \cap S_{2,c}$. As a result, the values in Z obtained in line 11, in the subroutine above, do not provide any information about c. One may think that an adversary could get some information about c by repeating values used for honest party input sets in prior RUN calls. However, this does not work since every new RUN call increments i which is used in the CH call, which means newly initialized keys (through line 7) are used in each RUN call. As $S_1 \cap S_{2,0} = S_1 \cap S_{2,1}$ for a query resulting in a non-$\bot$ response, the operations before line 10 in the response created above are always independent of the challenge bit. Thus, adversary A_{psi} can only guess the challenge bit. We get $\Pr[\, c' = c \mid b' = 0 \,] = 1/2$.

Combining with the above we get

$$\mathbf{Adv}^{\text{jit-oprf-2}}_{\mathsf{f},\Pi^{\text{jit}}}(\mathrm{A}_{\text{jit}}) = \Pr[\mathbf{G}^{\text{ini}}_{\mathsf{F},\Pi,2}(\mathrm{A}_{\text{psi}})] - \frac{1}{2}\ ,$$

which yields Eq. (7).

For resource usage, let $q = \mathrm{Q}^{\textsc{Run}}(\mathrm{A}_{\text{psi}})$. See that A_{jit} makes one TR oracle call for every RUN query made by A_{psi} and so $\mathrm{Q}^{\text{TR}}(\mathrm{A}_{\text{jit}}) = q$. Let $s_{i,j}$ be the size of party j set and $t_i = ts$ be the test set size output from $\mathsf{BS.Pg}$ in i-th RUN oracle query. Then, A_{jit} makes $t_i \cdot s_{i,2}$ CH oracle calls in i-th RUN query from A_{psi}. We get $\mathrm{Q}^{\text{CH}}(\mathrm{A}_{\text{jit}}) = \sum_{i=1}^{q} t_i \cdot s_{i,2} \leq t \cdot s_2$ where s_2 and t are upper bounds on the sum of the sizes of the sets of server (party 2) and sizes of bin allocation test sets, respectively, across all RUN queries. We have $\mathrm{Q}^{\text{RO}}(\mathrm{A}_{\text{jit}}) \leq \mathrm{Q}^{\text{RO}}(\mathrm{A}_{\text{psi}})$ due to the ROSIM subroutine defined above. ∎