X3DH with Deniable Authentication without Trusted Third Parties

Stanislaw Jarecki, Phillip Nazarian, and Apurva Rai

University of California, Irvine, USA
{sjarecki,pnazaria,apurvr1}@uci.edu

Abstract. Message Authentication in the Short Authenticated String model (SAS-MA) allows Alice and Bob to establish a secure channel without trust in any third party, as long as they can exchange *short authenticated strings*, e.g. 20 bits. In a recent paper, Gu et al. [17] showed a SAS-MA scheme based on Verifiable Random Functions (VRF), which can utilize the ephemeral keys sent in the X3DH Authenticated Key Exchange (AKE), allowing for extending X3DH to SAS-MA with minimal round complexity and no changes to X3DH key distribution. X3DH is used in many messaging apps, including WhatsApp and Signal, and a SAS-MA extension of X3DH would allow app users to authenticate their connections without trust in PKI or the app's Key Distribution Center (KDC), as long as they can exchange *short authenticated strings* (SAS), using out of band authenticated channels.
However, a major motivation behind using X3DH as an AKE is its *deniability* property, i.e. that an X3DH transcript cannot serve as a proof that either Alice or Bob established a secure connection with each other. The VRF-based SAS-MA extension of X3DH of [17] violates deniability, essentially because a VRF is a signature. We show an alternative SAS-MA scheme which offers the same ease of integration with X3DH as the VRF-based SAS-MA of [17], but it (almost) maintains the deniability of X3DH. The proposal is based on a *private* VRF (PVRF), which allows only 'designated-verifier' verification of correctness. We show a low-cost PVRF variant of ECVRF, and we show that X3DH extended by our PVRF-based SAS-MA adds human-centric no-trust-in-KDC authentication to X3DH while preserving the deniability properties of X3DH.

1 Introduction

Authenticated Key Exchange in the SAS model. Message Authentication (MA) in a Short Authenticated String model (SAS), introduced by Serge Vaudenay [34], allows two parties to (mutually) authenticate an arbitrary long message if the participants can exchange and compare *short authenticated strings*, e.g. of $d = 20$ bits. This form of authentication is *human centric*: short bitstrings can be encoded in user-friendly formats, e.g. as 6-digit decimal strings ($d = 20$), 6-character alphanumeric strings ($d = 30$), or 3-word English word sequences ($d = 42$). If Alice and Bob meet face-to-face then they can exchange these short

strings to establish security. Moreover, if the participants can trust the authenticity of each other's voices or video feeds, they can also exchange short messages over remote connections, although attacks on voice-based authentication using audio slicing and speech synthesis tools [29], show that remote voice-based authentication can only serve as a secondary authentication factor. Since authenticating a Key Exchange transcript implies an *Authenticated Key Exchange* (AKE), SAS-MA implies SAS-AKE, and allows parties to establish secure communication channels without pre-shared keys, passwords, or trust in third parties, e.g. a Key Certification Authority or a Key Distribution Service (KDC).

To see that achieving a SAS-MA scheme is not trivial, note that if the SAS string is a d-bit hash of the message, $\mathsf{sas} = H(m)$, then in time $2^{d/2}$ the attacker can find m, m' s.t. $H(m) = H(m')$, and if the sender adds a nonce r to the message then a man-in-the-middle attacker can use a precomputed table of size 2^d to find r' s.t. $H(r, m) = H(r', m')$. By contrast, a secure SAS-MA [34] ensures that if the two parties hold different messages $m \neq m'$ then even a man-in-the-middle attacker can make the verifier accept with at most 2^{-d} probability.[1] A series of works [15,34,22,23] showed SAS-MA schemes that use 3 protocol flows, which is minimal [34], and non-malleable commitments, achieving message *cross-authentication* (MCA) [24], i.e. authentication of both the initiator's and the responder's messages where the latter is not known beforehand.[2] Using such SAS-MCA with any 2-flow Key Exchange (KE), notably Diffie-Hellman [10], forms a 3-flow Authenticated Key Exchange in the SAS model (SAS-AKE).

X3DH and the nature of authentication it provides. An AKE scheme used to implement secure channels in several messaging apps, including WhatsApp and Signal, is X3DH [5], a variant of 'triple Diffie-Hellman' key exchange similar to HMQV [21]. Messaging apps use X3DH because of its several very attractive features: It is light, with only 4-5 exponentiations per party, it is as secure as Diffie-Hellman, and after the initiator retrieves the target counterparty's keys from the KDC administered by the app, a secure channel with the counterparty is established with a single protocol flow, allowing the initiator to encrypt/authenticate its first payload already with that single AKE flow.

However, authentication provided by an AKE like X3DH relies on trust that the KDC supplies each protocol party with the correct public key of their intended counterparty. One attempt to assure users that the KDC is providing correct keys are *Key Transparency* (KT) schemes [25,26], but KT schemes still require the user to trust the current hash of the KT data-structure, and to trust public auditors to verify the proofs stored in the logs of key dictionary updates.

[1] This is optimal because an attacker that follows the protocol still has 2^{-d} chance that the d-bit string expected by the verifier matches the one produced by the prover.

[2] The intuition for a 3-flow *uni-directional* SAS-MA scheme on which protocols of [15,34,22,23] are based, is that it is a generalization of a two-party coin-toss, i.e. the prover commits to (sas_0, m) for a random d-bit nonce sas_0, the receiver contributes a random d-bit nonce sas_1, and a short string $\mathsf{sas} = \mathsf{sas}_0 \oplus \mathsf{sas}_1$ is accepted by the receiver if the first-round commitment is opened to (sas_0, m).

SAS-AKE extension of X3DH. In a recent paper, Gu et al. [17] showed a SAS-MA scheme based on Verifiable Random Functions (VRF) [27], denoted VRF-SAS, with a novel property that VRF keys can be re-used across multiple instances, and after the initial VRF key distribution each VRF-SAS instance takes *only 2 protocol flows*. Further, VRF-SAS instantiated with ECVRF [16] can be integrated with X3DH so that the ECVRF keys used in VRF-SAS are the keys already present in X3DH, and VRF-SAS adds only 1 protocol flow to the X3DH handshake, which is minimal for bidirectional authentication in the SAS model. Consequently, with no changes to the X3DH key distribution process, and at low communication and computation costs, X3DH extended with VRF-SAS would allow the users of a messaging app to end-to-end authenticate the X3DH-established channel without trust in a third party, including the KDC.

Deniability of X3DH. However, the above list of X3DH benefits omits another motivating factor behind its choice in messaging apps, which is that it is *deniable*, i.e. that X3DH transcripts cannot serve as proofs, used to convince some third party, that an X3DH participant established a secure channel with its counterparty. The notion of Deniable AKE builds on deniable authentication of Dwork, Naor, and Sahai [12]. An AKE is deniable if Alice who intends to run an AKE with Bob should not worry about anyone, including the potentially dishonest Bob, acquiring a proof that could convince a third party distinguisher, Charlie, that Alice intended to execute an AKE with Bob. More technically, an AKE is deniable if any information an attacker can learn from an AKE interaction with Alice can be *simulated* solely from the attacker's own information.

Authentication/AKE deniability has been studied in many works, e.g. [12,9,14], with several considering specifically the deniability of X3DH and related protocols like HMQV, 3DH, or PQXDH, e.g. [33,13,20]. The strongest variant of this notion is *online* deniability, considered e.g. by [11,31,32], where the distinguisher Charlie can ask Bob to use Charlie-specified messages in an interaction with Alice. The work of [11] showed a 3-flow online deniable AKE (in the semi-adaptive setting where the attacker can compromise either party but not during the protocol execution) using non-committing encryption and *dual-receiver encryption*, where ciphertexts can be decrypted by either of two public keys. This deniable AKE was optimized in the works of [31,32], adding ring signatures to the above tools, who showed a 2-flow scheme that matches the communication pattern of X3DH, i.e. the first flow can be uploaded to the KDC as a *pre-key bundle* (see more on X3DH mechanics below), which achieves full on-line deniability for the initiator and a limited form of deniability for the responder.

X3DH does not achieve *online* deniability: If Charlie asks Bob to use Charlie's keys as Bob's ephemeral/one-time DH key contribution, then Bob cannot compute the resulting X3DH key, and thus cannot simulate an interaction with an honest Alice. However, the next-best property of *off-line deniability*, where the distinguisher Charlie sees only Bob's outputs after an interaction with Alice. X3DH was shown to satisfy offline deniability even against active attackers, in

the Generic Group Model (GGM) or under a 'knowledge of exponent' assumption implied by the Algebraic Group Model (AGM) [33,13,20].[3]

VRF-based SAS-AKE extension of X3DH breaks deniability. The SAS-AKE extension of X3DH via the VRF-SAS scheme of Gu et al. [17] preserves X3DH security but it violates its deniability properties. When VRF-SAS is integrated with X3DH, the short d-bit SAS token to be authenticated between Alice and Bob is derived from a VRF computed by the X3DH responder on the interaction transcript, using either her semi-static or one-time keys. (In addition, in VRF-SAS of [17] both parties use a VRF, but in the 2-party context the initiator's VRF can be replaced by a nonce.) A verifiable random function is a pseudorandom function with a non-interactive proof (NIZK) that $y = \mathsf{PRF}_k(x)$ is computed correctly given a public key vk which commits to k. VRF outputs computed on AKE transcripts yield secure SAS tokens because honest party's VRF outputs are unpredictable, but any key vk' which the attacker injects into the protocol binds the attacker to a SAS contribution it must provide on any AKE transcript. However, the NIZK provided by a VRF is effectively a signature on the transcript, which breaks deniability for the responder.

To see why VRF creates a deniability break in X3DH let us recall X3DH. In X3DH each party holds a long-term *identity key*, and it deposits this key at the KDC together with a *pre-key bundle* which includes a *semi-static key* and a batch of *one-time keys*, all of which are Diffie-Hellman KE contributions. The initiator Alice retrieves Bob's identity key, semi-static key, and a single unused one-time key if Bob's KDC-held bundle still includes them (the KDC erases each one-time key it sends out). The initiator then sends to the receiver a random *ephemeral key*, and the protocol then forks as follows: If Alice holds Bob's one-time key then the session key is derived by hashing *four* DH values which combine two Alice's DH contributions, her identity and ephemeral keys, with three Bob's DH contributions, his identity, semi-static, and one-time keys. If Bob's one-time keys run out, a case known as a *last resort*, then the protocol defaults to 3DH where Bob's semi-static key plays the role of his ephemeral DH contribution.

In the proposed X3DH and VRF-SAS integration [17], the responder issues a VRF using a one-time key in the default case, and a semi-static key in the last resort case. In the latter the responder loses its deniability unconditionally because its semi-static key can be retrieved from the KDC, signed by its identity key, but in the former case deniability is also lost if the distinguisher gains access to the KDC and sees the responder's pre-key bundle used in this instance.

Our Contribution: X3DH with SAS authentication *and* deniability. The deniability violation of X3DH extended by VRF-SAS of [17] comes from the use of VRFs which are publicly verifiable, like signatures. In this work we show a 2-party SAS-MA protocol which follows the VRF-based logic of the SAS-

[3] We note that some works on authentication/AKE deniability focus on *1-out-of-2* deniability, e.g. [19], where the distinguisher cannot tell if both Alice and Bob generate an AKE transcript, but it can learn that one of the two did. The above results on X3DH deniability correspond to a stronger *1-out-of-infinity* notion, where no one can prove that either Alice or Bob participated in an AKE instance.

MA of [17], but replaces a VRF with a *Privately Verifiable Random Function* (PVRF), which we introduce. Using our PVRF-based SAS-MA protocol denoted PVRF-SAS, for the SAS-AKE extension of X3DH, yields an X3DH extension that maintains the same efficiency and ease of adoption properties as [17] but in addition it (almost) maintains the same deniability properties as X3DH itself.

PVRF is an interactive 'designated verifier' VRF counterpart, involving a prover holding a PRF key k and a verifier holding the key commitment vk: The verifier first creates a one-time *verification token* vt together with the private verification state vts. The prover can then issue a proof π that PRF output $y = \mathsf{PRF}_k(x)$ is computed correctly under a key committed in vk, but the proof will be targeted to a specific token vt, and verifiable only by the party who holds the corresponding verification state vts. The proof system is sound, i.e. a cheating prover cannot create a proof that verifies on the wrong statement for an honestly created verification token vt. However, interaction with the prover is straight-line simulatable, and thus deniable, because tuple (vt, vts, π) can be efficiently sampled on the statement (vk, x, y) without the prover's witness k.

To keep the SAS-MA protocol at minimum round complexity we observe that the 2-party version of the VRF-SAS scheme of [17] remains secure if the initiator's SAS contribution is not computed via a VRF and is replaced by a nonce, and the SAS contribution of only the second party, the responder, is a verifiable PRF on the authenticated message. (The freedom this gives to the first party is not a threat, since it must decide its SAS contribution before the second party reveals its contribution.) Consequently, replacing the responder's VRF with a 2-flow PVRF keeps the protocol within 2 flows, which are minimal for the SAS model bidirectional authentication. We call the resulting SAS-MA scheme PVRF-SAS.

Deniability of the proposed SAS-AKE X3DH extension. Our end-goal is protocol X3DH + PVRF-SAS, i.e. a SAS-AKE extension of X3DH, with the same security, efficiency, and ease of adoption properties as the proposal in [17], but which would maintain the same deniability properties as X3DH. We achieve this objective with a caveat. Table 1 shows a comparison of the deniability properties, as defined by Katsumata et al. [20], achieved (1) by X3DH itself, [20], (2) by the X3DH SAS-AKE extension 'X3DH+VRF-SAS' proposed by [17], and (3) by the X3DH SAS-AKE extension 'X3DH+PVRF-SAS' proposed in this work.

The five rows in Table 1 correspond to 5 deniability attack scenarios: $\mathcal{H}$ and $\mathcal{C}$ denote the sets of resp. honest and corrupted parties, and (s, r) denote resp. the 'sender' and 'receiver' parties of X3DH, i.e. resp. the initiator and the responder. Using this terminology, $(s, r) \in \mathcal{H} \times \mathcal{C}$ is a case of an honest initiator/sender whose intended counterparty is corrupt and attempts to gain a proof of the initiator's engagement in the protocol. Similarly, $(s, r) \in \mathcal{C} \times \mathcal{H}$ is the case of an honest receiver/responder whose intended counterparty is corrupt and attempts to gain a proof of the responder's engagement in the protocol. Finally, $(s, r) \in \mathcal{H} \times \mathcal{H}$ considers the deniability of two honest interacting parties and whether their transcript can be efficiently simulatable from public information. The last two cases are split into two sub-cases, where $\mathsf{otpk} \neq \perp$ stands for the default case

	X3DH	X3DH+VRF-SAS	X3DH+PVRF-SAS
$(s,r) \in \mathcal{H} \times \mathcal{C}$	high	high	high
$(s,r) \in \mathcal{C} \times \mathcal{H}$, $\mathsf{otpk} \neq \perp$	high	none/high*	high
$(s,r) \in \mathcal{C} \times \mathcal{H}$, $\mathsf{otpk} = \perp$	high	none	medium
$(s,r) \in \mathcal{H} \times \mathcal{H}$, $\mathsf{otpk} \neq \perp$	high	none/high*	high
$(s,r) \in \mathcal{H} \times \mathcal{H}$, $\mathsf{otpk} = \perp$	medium	none	medium

Table 1: Deniability of protocols X3DH, X3DH + VRF-SAS [17], and X3DH+PVRF-SAS [this work]. This table lists the highest level of leakage for which deniability holds. In all cases, deniability is achieved for high disclosure and KDC leakage, except * indicates cases deniability given no KDC leakage.

where the initiator retrieves the responder's one-time pre-key from the KDC (see above), and $\mathsf{otpk} = \perp$ is the *last resort* case where the KDC runs out of the responder's one-time pre-keys. The high and medium deniability levels correspond to the amount of honest party information which is leaked to the distinguisher: In both cases the distinguisher learns the private key corresponding to the honest party's long-term identity key, but in the high level the distinguisher also learns their dynamic state, including their secret semi-static key and any *unused* private one-time pre-keys. Finally, we assume 'KDC leakage' by default, i.e. that the distinguisher learns the accused party's full pre-key bundle, and the two entries which assume 'no KDC leakage' are marked with an asterisk sign *.

All deniability properties listed are achieved in the face of both honest-but-curious and malicious adversaries, except the cryptographic assumptions differ between the Random Oracle Model (ROM) for the former and the ROM + the Algebraic Group Model (AGM) for the latter. As can be seen from the table, our proposal X3DH+PVRF-SAS maintains the deniability properties of X3DH except in the case where the receiver responds in the last resort case, i.e. when its store of one-time keys in the pre-key bundle is exhausted, and, furthermore, the receiver's private semi-static key leaks to the distinguisher. Note that X3DH failed to fully protect receiver's deniability in this case as well, but in the $(s,r) \in \mathcal{H} \times \mathcal{H}$ case where the protocol transcript can frame either the sender or the receiver but not both, whereas in X3DH+PVRF-SAS the receiver in this case would be personally un-deniable if it engaged with a corrupted counterparty. This appears a modest degradation of deniability compared to X3DH, and a moderate price for extending X3DH to SAS-AKE, i.e. achieving human-centric secure channel authentication without trust in third parties like KDC.

Key Building Block: Efficient UC PVRF from ECVRF and SPHF's. We model PVRF in a Universally Composable (UC) model [4]. We base the UC PVRF functionality on the UC VRF functionality [8,1,2], amending it to the setting of a single target verifier. A universally composable PVRF is easier to use as a subprotocol in applications, as exemplified by the SAS-MA scheme PVRF-SAS. Scheme PVRF-SAS relies on the pseudorandomness, simulatability, output uniqueness, and soundness properties of PVRF, and since the

same PVRF key is reused in multiple SAS-MA instances, the SAS-MA security argument inherently considers a composition of PVRF instances. The UC PVRF model captures all these properties, and a PVRF that realizes it can be substituted with an ideal PVRF functionality in the PVRF-SAS security proof, which makes that proof clearer and easier to inspect.

We construct an efficient UC PVRF scheme, called PECVRF, as a privately verifiable version of ECVRF [16], which is a well-known VRF based on the Diffie-Hellman function. Scheme PECVRF uses the same underlying Diffie-Hellman PRF as ECVRF, i.e. $vk = g^k$ and $\mathsf{PRF}_k(x) = \mathsf{Hash}(vk, x)^k$ (we simplify this overview by omitting the outer hash applied to this value), but it replaces the NIZK that (g, vk, g_x, y) is a DH tuple for $g_x = \mathsf{Hash}(vk, x)$, with a well-known Smooth Projective Hash Function (SPHF) [7] for the same language, i.e. the verifier picks a pair of random field elements (α, β) and sends $h = g^\alpha \cdot g_x^\beta$ to the prover, who responds with $\pi = h^k$, which the verifier accepts only if $\pi = vk^\alpha \cdot y^\beta$. This satisfies the requirement of prover deniability, because tuple (α, β, h, π) can be sampled given only the statement (g, vk, g_x, y).

However, an SPHF is not a zero-knowledge proof, and indeed in the above system the prover exposes an exponentiation oracle $(\cdot)^k$ which could be used e.g. to evaluate the PVRF on $x' \neq x$. To fix this, we turn this SPHF into a *designated verifier* zero-knowledge proof [19] by adding a simulation trapdoor $h' = g_1^\alpha \cdot g_2^\beta$, where $g_i = \mathsf{Hash}_i(vk, x)$, and forming the verification token as $vt = (h, h', \tau)$ where τ is a NIZK that (h, h') are computed correctly using some α, β. The real-world prover outputs $\pi = h^k$ only if the prover's NIZK τ verifies, and this enables simulation because under the DDH assumption the simulator can embed correlated trapdoored responses (vk^r, y^r) into (g_1, g_2) pairs output by $\mathsf{Hash}_1, \mathsf{Hash}_2$, and compute π as $\pi = (h')^{1/r}$.[4]

Organization. We recall the standard definitions of the Computational (CDH), Decisional (DDH), and Gap (GapDH) Diffie-Hellman assumptions in Appendix A. In Section 2 we define PVRF and show our PVRF scheme based on ECVRF. In Section 3 we define SAS-MA and we specify a SAS-MA scheme based on PVRF. In Section 4 we recall a definition of Deniable AKE, and in Section 5 we show an integration of our PVRF-based SAS-MA with X3DH and argue that it is a secure *and deniable* AKE. For the reasons of space constraints much material, including security models and proofs, is deferred to the appendices.

[4] One could also implement vt as a one-time public key amended by a NIZK proof of knowledge of the corresponding private key, and replace the prover's NIZK with a Designated-Verifier NIZK (DV-NIZK) proof of knowledge [19] or a 'disjunctive' Non-Interactive Witness Indistinguishable (NIWI) proof [6], where the prover proves that either (g, vk, g_x, y) is a DH tuple or that the prover holds the secret key corresponding to vt. Constructions using DV-NIZK proofs have been used to implement deniable authentication, e.g. [32,14], hence PVRF constructed using these techniques probably passes a corresponding notion of PVRF deniability. Our SPHF-based PECVRF has roughly the same costs as ECVRF using DV-NIZK as above, but it offers a performance trade-off of 4 (multi)-exps for the verifier and 2 for the prover, while a DV-NIZK implementation using a disjunctive proof [6] uses 3 (multi-)exps per party.

2 Privately-Verifiable Random Functions

A Verifiable Random Function (VRF), introduced by Micali, Rabin, and Vadhan [27], is a Pseudorandom Function (PRF) extended by the presence of a verification key vk that is created by the key generation algorithm along with the PRF key k. The VRF evaluation algorithm computes the PRF output z along with a noninteractive proof π that $z = \mathsf{PRF}_k(x)$, where PRF is the underlying PRF and k is the key committed to by verification key vk. Any verifier holding vk can verify this proof. Crucially, the function remains pseudorandom in the view of parties that have seen vk. VRFs must also satisfy soundness, meaning that it is infeasible to create a verification key vk and proofs π, π' that two different values z, z' are both correct VRF outputs on the same argument x.

A VRF proof acts like a signature on message m under key vk, and the use of VRFs in the SAS-MA protocol of [17] is therefore deleterious to deniability. In this work, we consider the replacement of VRFs with *Privately-Verifiable Random Functions* (PVRFs), which behave much like VRFs except that their proofs are only verifiable by a single target verifier who has created a verification token vt. As explained in Section 1, this notion is inspired by the verification properties of SPHFs and DV-NIZKs.

Formally, we define a PVRF as a tuple of the following 4 algorithms:

- $\mathsf{KeyGen}(1^\kappa) \to (k, vk)$, used by the prover to generate a verification key vk with corresponding secret key k.
- $\mathsf{PreVerify}(vk, x) \to (vt, vts)$, used by a verifier of a PRF evaluation on argument x under key vk, generating the verification token vt and state vts.
- $\mathsf{Eval}(k, x, vt) \to (z, \pi)$, used by the prover to evaluate the PRF on argument x under key k, generating output z and proof π which targets token vt.
- $\mathsf{Verify}(vk, x, vts, z, \pi) \to 0/1$, used by the verifier holding state vts to verify proof π that z is the correct PRF output on argument x under key vk.

Figure 1 is protocol PECVRF, a privately-verifiable variant of the ECVRF protocol. As discussed in the Introduction, PECVRF computes the same pseudorandom function as ECVRF, but it achieves private (and deniable) verifiability using smooth projective hash functions (SPHFs) instead of non-interactive zero-knowledge proofs (NIZKs). However, for simulatability of the prover's SPHF response the verification token created by the verifier includes a simulation-enabling trapdoor together with a NIZK proof that it is computed correctly. The figure has been written in a way that mirrors its ideal functionality in the UC model and thus also takes a unique pointer kptr as input to KeyGen.

In Appendix B we define a UC model for PVRF and prove that it is realized by PECVRF. This functionality, however, only ensures PVRF security, i.e. correctness, pseudorandomness, and soundness. These properties are sufficient to prove the security of our SAS-MA scheme PVRF-SAS (see Section 3), but not to prove the deniability of the key-reusing composition of PVRF-SAS with X3DH. We therefore provide a separate, game-based definition for PVRF *deniability* in Figure 2. PVRF deniability has two variants: standard, defined by game $\mathsf{Game}^{\mathsf{Den}}_{\mathsf{D,PVRF}}$, and strong, defined by game $\mathsf{Game}^{\mathsf{SDen}}_{\mathsf{D,PVRF}}$. Both games involve

$\mathbb{G}$ is a group of prime-order p with generator g, and H_i, H_o, H_a, H_b, and H_{FS} are hash functions whose codomains are respectively $\mathbb{G}$, $Z := \{0,1\}^d$, $\mathbb{G}$, $\mathbb{G}$, and $\mathbb{Z}_p$.

On $(\mathsf{KeyGen}, \mathsf{kptr})$, P does:

- sample $k \leftarrow_\$ \mathbb{Z}_p$, set $vk := g^k$, and store (kptr, k, vk)
- output $(\mathsf{VerificationKey}, \mathsf{kptr}, \mathsf{P}, vk)$

On $(\mathsf{PreVerify}, vk, x)$, V does:

- sample $\alpha, \beta \leftarrow_\$ \mathbb{Z}_p$, set $h := g^\alpha \cdot \mathsf{H}_i(vk,x)^\beta$ and $h' := \mathsf{H}_a(vk,x)^\alpha \cdot \mathsf{H}_b(vk,x)^\beta$
- sample $r_1, r_2 \leftarrow_\$ \mathbb{Z}_p$, and set $\eta := g^{r_1} \cdot \mathsf{H}_i(vk,x)^{r_2}$, $\eta' := \mathsf{H}_a(vk,x)^{r_1} \cdot \mathsf{H}_b(vk,x)^{r_2}$, $c := \mathsf{H}_{\mathsf{FS}}(vk, x, h, h', \eta, \eta')$, $s := (r_1 - c\cdot\alpha, r_2 - c\cdot\beta)$, and $\tau = (c, s)$
- set $vt := (h, h', \tau)$, $vts := (\alpha, \beta)$ and store (vt, vk, x, vts)
- output $(\mathsf{VerificationToken}, vt)$

On $(\mathsf{Eval}, \mathsf{kptr}, x, vt)$, P retrieves (kptr, k, vk) and does:

- parse $(h, h', \tau) \leftarrow vt$ and $(c, (s_1, s_2)) \leftarrow \tau$
- set $\eta := g^{s_1} \cdot \mathsf{H}_i(vk,x)^{s_2} \cdot h^c$ and $\eta' := \mathsf{H}_a(vk,x)^{s_1} \cdot \mathsf{H}_b(vk,x)^{s_2} \cdot h'^c$, and if $c \neq \mathsf{H}_{\mathsf{FS}}(vk, x, h, h', \eta, \eta')$, then abort
- set $w := \mathsf{H}_i(vk,x)^k$, $z := \mathsf{H}_o(w)$, $v := h^k$, and $\pi := (w, v)$
- output $(\mathsf{Evaluated}, vk, x, vt, z, \pi)$

On $(\mathsf{Verify}, vk, x, vt, z, \pi)$, V retrieves (vt, vk, x, vts) and does:

- parse $(w, v) \leftarrow \pi$ and $(\alpha, \beta) \leftarrow vts$
- set $f \leftarrow 1$ if $z = \mathsf{H}_o(w)$ and $v = vk^\alpha \cdot w^\beta$; otherwise, set $f \leftarrow 0$
- output $(\mathsf{Verified}, vk, x, vt, z, \pi, f)$

Fig. 1: Protocol PECVRF: the *private* elliptic curve verifiable random function

an algorithm D who seeks to distinguish between real PVRF outputs generated by an honest prover and "faked" outputs generated by a simulator $\mathsf{FakeEval}$. In the standard deniability game the real prover executes on a verification token vt that is honestly generated by $\mathsf{PreVerify}$, and $\mathsf{FakeEval}$ takes as input the corresponding secret vts. In the strong deniability game, there is an active adversary $\mathcal{A}$ that maliciously generates vt, and $\mathsf{FakeEval}$ is allowed black-box access to $\mathcal{A}$. Protocol PVRF is *deniable* if there exists an efficient algorithm $\mathsf{FakeEval}$ and a negligible function ε s.t. for all efficient D $\Pr[1 \leftarrow \mathsf{Game}^{\mathsf{Den}}_{\mathsf{D},\mathsf{PVRF}}] \leq \frac{1}{2} + \varepsilon(\kappa)$. Protocol PVRF is *strongly deniable* if for all efficient algorithms $\mathcal{A}$ there exists an efficient algorithm $\mathsf{FakeEval}_{\mathcal{A}}$ and a negligible function ε s.t. for all efficient D $\Pr[1 \leftarrow \mathsf{Game}^{\mathsf{SDen}}_{\mathsf{D},\mathcal{A},\mathsf{PVRF}}] \leq \frac{1}{2} + \varepsilon(\kappa)$.

Theorem 1. PECVRF *is* deniable *if the* DDH *problem is hard in* $\mathbb{G}$ *and* H_i *is modeled as a random oracle.*

Theorem 2. PECVRF *is* strongly deniable *in the AGM if the* DDH *problem is hard in* $\mathbb{G}$ *and* H_i *is modeled as a random oracle.*

We defer the proofs of Theorems 1 and 2 to Appendix B.

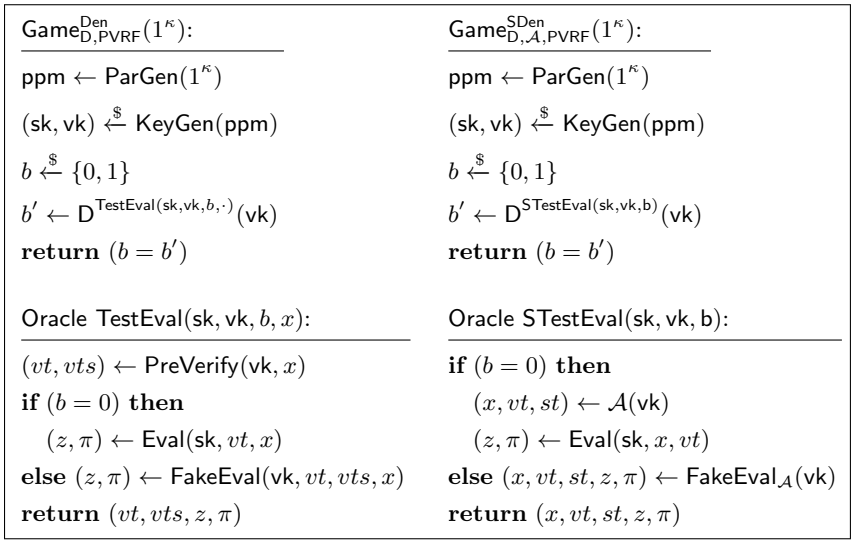

Fig. 2: PVRF deniability game, with standard and strong variants.

3 Message Authentication in the SAS Model

In this section we recall the syntax and the security goals of Message Authentication in the Short Authenticated String model (SAS-MA), and we show a SAS-MA scheme based on Private VRF (PVRF), introduced in Section 2. Our SAS-MA scheme requires only two flows after key generation, but unlike the VRF-based SAS-MA of [17] it does not create a publicly verifiable transcript, and hence it can be used in conjunction with a *deniable* Authenticated Key Exchange protocol, as we show in Section 5.

SAS-MA was first defined by Vaudenay [34], but we follow [17] in adjusting SAS-MA syntax for schemes in which each party generates a public key which it can re-use in multiple SAS-MA protocol instances. Unlike [17], we restrict our definition for the case of only 2 parties, P_1 and P_2, who want to mutually authenticate a message to each other. Their respective inputs are denoted m_1 and m_2, and they want to confirm that $m_1 = m_2$. The parties communicate over insecure channels and have no shared source of trust, except for an additional communication resource of a low-bandwidth bi-directional authenticated channel. Namely, for some parameter d, each P_i can send one d-bit message, denoted sas_i, on the *Short Authenticated String* (SAS) *channel* connecting P_i and its counterparty. This channel is not secret, so the adversary learns sas_i, but it preserves integrity and source authentication, i.e. the adversary cannot modify this message or replace it with its own. In more detail, we define a SAS-MA scheme as a following algorithm tuple:

– $\mathsf{PreProc}(1^\kappa) \to (k, vk)$, a pre-processing algorithm used by P_2 to generate a re-usable public key vk and a corresponding private key k.

– $\mathsf{Init}(vk, m) \to (pm_1, \mathsf{st})$, initiator P_1's generation of a protocol message and an ephemeral state, given counterparty's vk and an authentication target m.
– $\mathsf{Resp}(k, m, pm_1) \to (pm_2, \mathsf{sas}_2)$, responder P_2's generation of a protocol message and a SAS value, given its private key k and a target message m.
– $\mathsf{InitFin}(\mathsf{st}, pm_2) \to \mathsf{sas}_1$, the initiator P_1's generation of its SAS value.

Finally, each P_i sends its SAS value $\mathsf{sas}_i \in \{0,1\}^d$ on its SAS channel to the counterparty, and each party accepts if and only if $\mathsf{sas}_1 = \mathsf{sas}_2$. Figure 3 shows PVRF-SAS, our PVRF-based SAS-MA scheme. In this scheme the short authentication string $\mathsf{sas} \in \{0,1\}^d$ is formed as $z \oplus \mathsf{nonce}$, where z is the output of responder P_2's PVRF on input m, and nonce is a random string chosen by P_1.

$\mathsf{PVRF} = (\mathsf{KeyGen}, \mathsf{PreVerify}, \mathsf{Eval}, \mathsf{Verify})$ is a PVRF with range $\{0,1\}^d$.

$\underline{\mathsf{PreProc}(1^\kappa)}$:
$(k, vk) \leftarrow \mathsf{KeyGen}(1^\kappa)$
return (k, vk)

$\underline{\mathsf{Init}(vk, m)}$:
$\mathsf{nonce} \leftarrow_\$ \{0,1\}^d$
$(vt, vts) \leftarrow \mathsf{PreVerify}(vk, m)$
return $((vt, \mathsf{nonce}), (vk, m, \mathsf{nonce}, vts))$

$\underline{\mathsf{Resp}(k, m, pm_1 = (vt, \mathsf{nonce}))}$:
$(z, \pi) \leftarrow \mathsf{Eval}(k, vt, m)$
return $((z, \pi), z \oplus \mathsf{nonce})$

$\underline{\mathsf{InitFin}(\mathsf{st} = (vk, m, \mathsf{nonce}, vts), pm_2 = (z, \pi))}$:
assert $\mathsf{Verify}(vk, vts, m, z, \pi)$
return $z \oplus \mathsf{nonce}$

Fig. 3: SAS-MA scheme PVRF-SAS

Security of PVRF-SAS follows from the properties of the underlying PVRF. In particular, PVRF pseudorandomness (and input m freshness) ensures that z is effectively random until after adversary $\mathcal{A}$ commits to nonce by executing Resp. Clearly nonce is random until $\mathcal{A}$ executes Init, and PVRF soundness ensures that z is effectively fixed by $\mathcal{A}$ at that time. Thus, neither party's sas can be predicted and controlled by a man-in-the-middle adversary, and $\mathcal{A}$ has no attack that succeeds with probability better than $1/2^d$. We defer to Appendix C the formal definition of SAS-MA security and the proof that PVRF-SAS achieves it. This proof is extremely similar to that of [17]; recall that we replace VRF with PVRF in order to strengthen deniability—not security—properties.

4 Deniable Authenticated Key Exchange

To model the security properties of X3DH, Hashimoto, Katsumata, and Wiggers [18] introduced the notion of Bundled Authenticated Key Exchange (BAKE), which extends the standard definition of AKE with *prekey bundles*. Each party in BAKE creates a bundle containing some number L of one-time prekeys, and BAKE key exchange is an asymmetric interaction between a *sender* and *receiver*, who must agree on an index $t \in [L]$ in the receiver's prekey bundle. The BAKE

security model enforces that each prekey is only used once, but BAKE also allows for a fallback mechanism designated $t = \perp$ whereby a special *last-resort prekey* is used instead, in case the receiver's bundle has run out of one-time keys. In that eventuality, the last-resort key (also called the *semi-static key*) might be reused across many key exchanges, but the forward security properties of these key exchanges are diminished compared to those that use one-time keys.

Katsumata et al. [20] later defined *deniability* (with various gradations) for BAKE protocols, but in order to use their models we must make some adaptations. Adding SAS-MA authentication to X3DH involves an extra communication flow from the receiver to the sender. A *modified* bundled authenticated key exchange $\mathsf{BAKE}^\dagger$ is a tuple of algorithms ($\mathsf{IdKeyGen}$, $\mathsf{PreKeyBundleGen}$, Send, $\mathsf{Receive}$, $\mathsf{SenderReceive}$), parameterized by bundle size L, with the following syntax:

- $\mathsf{BAKE}^\dagger.\mathsf{IdKeyGen}(1^\kappa) \to (\mathsf{ipk}, \mathsf{isk})$, an identity key generation algorithm that outputs an identity public key ipk and corresponding secret key isk.
- $\mathsf{BAKE}^\dagger.\mathsf{PreKeyBundleGen}(\mathsf{isk}_u) \to (\mathbf{prek}_u, \mathsf{st}_u)$, a prekey bundle generation algorithm that, on input user u's identity secret key, outputs a bundle of L prekeys (and a last-resort prekey) $\mathbf{prek}_u = (\mathsf{prek}_{u,t})_{t \in [L] \cup \{\perp\}}$, and a user state st_u. (The state may for example include the secret keys associated to the public keys in $\mathbf{prek}_u$.)
- $\mathsf{BAKE}^\dagger.\mathsf{Send}(\mathsf{isk}_s, \mathsf{ipk}_r, \mathsf{prek}_{r,t}) \to (K, \rho, \mathsf{temp})$, a sender algorithm that, on input sender s's identity secret key isk_s and the intended receiver r's identity key ipk_r and prekey $\mathsf{prek}_{r,t}$, outputs a key K, a handshake message ρ, and a temporary state temp for the sender.
- $\mathsf{BAKE}^\dagger.\mathsf{Receive}(\mathsf{isk}_r, \mathsf{st}_r, \mathsf{ipk}_s, t, \rho) \to (K', m, \mathsf{sas}_r, \mathsf{st}_r)$, a receiver algorithm that, on input receiver r's identity secret key isk_r and state st_r, sender s's identity key ipk_s, prekey index $t \in [L] \cup \{\perp\}$, and handshake message ρ, outputs a key K', a response message m, a short authentication string sas_r, and an updated state st_r. (Key agreement may fail, in which case $K' = \perp$.)
- $\mathsf{BAKE}^\dagger.\mathsf{SenderReceive}(\mathsf{temp}, m) \to \mathsf{sas}_s$, a sender completion algorithm that, on input state temp from Send and a response message m, outputs a short authentication strings sas_r. (Authentication may fail, in which case $\mathsf{sas}_s = \perp$.)

We highlight the differences between BAKE and $\mathsf{BAKE}^\dagger$ in yellow text. The syntax of $\mathsf{IdKeyGen}$ and $\mathsf{PreKeyBundleGen}$ is unchanged. The sender saves a temporary state temp, awaits a response message m output from $\mathsf{Receive}$, and then uses $\mathsf{SenderReceive}$ to complete SAS authentication. Note that K is output as expected in BAKE algorithms, and out-of-band verification of $\mathsf{sas}_r = \mathsf{sas}_s$ is only an 'add-on' to the regular BAKE execution.

4.1 Deniability Against Honest-but-Curious Accusers

Figures 4 and 5 are the deniability game $\mathsf{Game}^{\mathsf{ATK}}_{\mathsf{D},\mathcal{H},\mathcal{L}_{\mathsf{leak}},\mathcal{D}_{\mathsf{disc}},\mathcal{L}_{\mathsf{KDC}}}$ for honest-but-curious accusers (parameterized by $\mathsf{leak}, \mathsf{disc}, \mathcal{L}_{\mathsf{KDC}}$, and ATK, explained below), with differences from [20] marked in yellow text. The game concerns a *distinguisher* D (also called a judge) attempting to distinguish between real and

$\mathsf{Game}^{\mathsf{ATK}}_{\mathsf{D},\mathcal{H},\mathcal{L}_{\mathsf{leak}},\mathcal{D}_{\mathsf{disc}},\mathcal{L}_{\mathsf{KDC}}}(1^\kappa, \mathsf{mode})$:

$\mathcal{C} := \mathcal{N} \setminus \mathcal{H}$

for user $u \in \mathcal{N}$ **do**

  $(\mathsf{ipk}_u, \mathsf{isk}_u) \xleftarrow{\$} \mathsf{BAKE}^\dagger.\mathsf{IdKeyGen}(1^\kappa)$

  $\mathsf{counter}_u := 0$

  if $(u \in \mathcal{H})$ **then**

    $(\mathbf{prek}_u, \mathsf{st}_u) \xleftarrow{\$} \mathsf{BAKE}^\dagger.\mathsf{PreKeyBundleGen}(\mathsf{isk}_u)$

if $(\mathsf{mode} = \mathsf{real})$ **then**

  for user $u \in \mathcal{C}$ **do**

    $(\mathbf{prek}_u, \mathsf{st}_u) \xleftarrow{\$} \mathsf{BAKE}^\dagger.\mathsf{PreKeyBundleGen}(\mathsf{isk}_u)$

    $\mathsf{st}^{\mathsf{init}}_u := \mathsf{st}_u$

else if $(\mathsf{mode} = \mathsf{sim})$ **then**

  $((\mathbf{prek}^*_u)_{u\in\mathcal{H}}, (\mathbf{prek}_u)_{u\in\mathcal{C}}, \mathsf{st}_{\mathsf{Sim}}) \xleftarrow{\$} \mathsf{SimPreK}\,((\mathsf{ipk}_u)_{u\in\mathcal{N}}, (\mathsf{isk}_u)_{u\in\mathcal{C}}, (\mathbf{prek}_u)_{u\in\mathcal{H}})$

  if $\neg\mathcal{L}_{\mathsf{KDC}}$ **then** $(\mathbf{prek}_u)_{u\in\mathcal{H}} \leftarrow (\mathbf{prek}^*_u)_{u\in\mathcal{H}}$

$\mathsf{st}_{\mathsf{D}} \xleftarrow{\$} \mathsf{D}^{\mathcal{O}_{\mathsf{ATK}}}((\mathsf{ipk}_u, \mathbf{prek}_u)_{u\in\mathcal{N}})$

if $(\mathsf{mode} = \mathsf{sim})$ **then**

  $(\mathsf{st}_u, \mathsf{st}^{\mathsf{init}}_u) \xleftarrow{\$} \mathsf{SimSt}_{\mathcal{C}}(\mathsf{st}_{\mathsf{Sim}})$

$\mathsf{leak}_{\mathsf{D}} := \mathcal{L}_{\mathsf{leak}}((\mathsf{isk}_u, \mathsf{st}_u)_{u\in\mathcal{H}})$

$\mathsf{disc}_{\mathsf{D}} := \mathcal{D}_{\mathsf{disc}}((\mathsf{isk}_u, \mathsf{st}_u, \mathsf{st}^{\mathsf{init}}_u)_{u\in\mathcal{C}})$

$b \xleftarrow{\$} \mathsf{D}(\mathsf{st}_{\mathsf{D}}, \mathsf{leak}_{\mathsf{D}}, \mathsf{disc}_{\mathsf{D}})$

return b

Fig. 4: Local and global $\mathsf{BAKE}^\dagger$ deniability game for *honest-but-curious* accusers, with changes relative to BAKE deniability [20] marked in yellow.

simulated transcripts of $\mathsf{BAKE}^\dagger$ key exchanges between N parties. Some subset $\mathcal{H} \subseteq [N]$ of the parties are honest and wish to deny their $\mathsf{BAKE}^\dagger$ interactions. The rest $\mathcal{C} = [N] \setminus \mathcal{H}$ are *accusers* who wish to incriminate the honest parties.

The deniability game runs either in $\mathsf{mode} = \mathsf{real}$ or $\mathsf{mode} = \mathsf{sim}$. Distinguisher D is given the public keys of all parties and access to an oracle that adaptively directs pairs $(s, r) \in [N] \times [N]$ of parties to perform key exchanges. In the real mode, the real $\mathsf{BAKE}^\dagger$ algorithms are used to generate a transcript and return it to D. In the sim mode, there is instead a simulator algorithm $\mathsf{SimTrans}$ that generates the returned transcript. The simulator is given the secrets of accusers, but not those of honest parties. Intuitively, deniability is implied if there exists such a simulator that can produce transcripts indistinguishable from those of real interactions. The game starts with each party running the $\mathsf{IdKeyGen}$ and $\mathsf{PreKeyBundleGen}$ algorithms. We follow the choice of [20] to assume that all

Oracle $\mathcal{O}_{\mathsf{ATK}}(s, r)$:

assert $((s, r) \in \mathcal{N} \times \mathcal{N}) \wedge (s \neq r) \wedge ((s, r) \notin \mathcal{C} \times \mathcal{C})$
if $(\mathsf{ATK} = \mathsf{local})$ **then assert** $(s, r) \notin \mathcal{H} \times \mathcal{H}$
$\mathsf{counter}_r \leftarrow \mathsf{counter}_r + 1,\ t := \mathsf{counter}_r$
if $(t > L)$ **then** $t \leftarrow \perp$
if $(\mathsf{mode} = \mathsf{real})$ **then**
  $(K, \rho, \boxed{\mathsf{temp}}) \xleftarrow{\$} \mathsf{BAKE}^{\dagger}.\mathsf{Send}(\mathsf{isk}_s, \mathsf{ipk}_r, \mathsf{prek}_{r,t})$
  $(K', \boxed{m, \mathsf{sas}_r,} \mathsf{st}_r) \xleftarrow{\$} \mathsf{BAKE}^{\dagger}.\mathsf{Receive}(\mathsf{isk}_r, \mathsf{st}_r, \mathsf{ipk}_s, t, \rho)$
  $\boxed{\mathsf{sas}_s \leftarrow \mathsf{BAKE}^{\dagger}.\mathsf{SenderReceive}(\mathsf{temp}, m)}$
else if $(\mathsf{mode} = \mathsf{sim})$ **then**
  if $(s, r) \in \mathcal{H} \times \mathcal{C}$ **then**
    $(K, \rho, \boxed{m, \mathsf{sas}_s}, \mathsf{st}_{\mathsf{Sim}}) \xleftarrow{\$} \mathsf{SimTrans}(\mathsf{isk}_r, \mathsf{st}_{\mathsf{Sim}}, (s, r, t))$
  else if $(s, r) \in \mathcal{C} \times \mathcal{H}$ **then**
    $(K', \rho, \boxed{m, \mathsf{sas}_r}, \mathsf{st}_{\mathsf{Sim}}) \xleftarrow{\$} \mathsf{SimTrans}(\mathsf{isk}_s, \mathsf{st}_{\mathsf{Sim}}, (s, r, t))$
    $\mathsf{st}_r \xleftarrow{\$} \mathsf{SimSt}_{\mathcal{H}}(\mathsf{isk}_r, \mathsf{st}_r)$
  else if $(s, r) \in \mathcal{H} \times \mathcal{H}$ **then**
    $(K, K', \rho, \boxed{m, \mathsf{sas}_s, \mathsf{sas}_r}, \mathsf{st}_{\mathsf{Sim}}) \xleftarrow{\$} \mathsf{SimTrans}(\perp, \mathsf{st}_{\mathsf{Sim}}, (s, r, t))$
    $\mathsf{st}_r \xleftarrow{\$} \mathsf{SimSt}_{\mathcal{H}}(\mathsf{isk}_r, \mathsf{st}_r)$
if $(s, r) \in \mathcal{H} \times \mathcal{C}$ **then return** $(K, \rho, \boxed{m, \mathsf{sas}_s})$
else if $(s, r) \in \mathcal{C} \times \mathcal{H}$ **then return** $(K', \rho, \boxed{m, \mathsf{sas}_r})$
else return $(K, K', \rho, \boxed{m, \mathsf{sas}_s, \mathsf{sas}_r})$

Fig. 5: Oracle $\mathcal{O}_{\mathsf{ATK}}$ for honest-but-curious accusers in $\mathsf{BAKE}^{\dagger}$ deniability game in Fig. 4, with changes from BAKE deniability game [20] marked in yellow.

parties' identity keys are honestly generated, even in the simulated world. On the other hand, accusers' simulated-world prekey bundles are generated by the simulator algorithm SimPreK, which is given as input all honest parties' public keys, prekey bundles and the identity secret keys of the accusers.

Leakage and Disclosure. In the deniability model of [20], the simulator is also allowed to overwrite all honest parties' prekey bundles. It is these simulated prekey bundles that are then given to D, and the honestly-generated ones are discarded. Effectively, such a model only implies deniability against a judge who cannot observe the prekey bundles generated by honest parties. This may be a reasonable model for some scenarios, but one can also imagine a judge capable of monitoring (or retroactively seizing) the prekey bundles saved by the KDC. To capture the stronger deniability notion of this setting, we parameterize the deni-

ability game by a bit $\mathcal{L}_{\mathsf{KDC}}$, which represents whether honestly-generated prekey bundles are "leaked" to D. If $\mathcal{L}_{\mathsf{KDC}} = 0$ then these bundles can be overwritten by $\mathsf{SimPreK}$ as in [20], and if $\mathcal{L}_{\mathsf{KDC}} = 1$, then this overwriting is skipped. As shown in Section 1, there are schemes whose deniability is severely diminished when this overwriting is disallowed by $\mathcal{L}_{\mathsf{KDC}} = 1$.

At the end of the deniability game (i.e. after all key exchanges), D is allowed some final secret information before attempting to distinguish the real and simulated worlds. This information is classified as *leakage* from honest parties and *disclosure* from accusers, and the game is parameterized by leakage level leak and disclosure level disc, both in $\{\mathsf{low}, \mathsf{med}, \mathsf{high}\}$. At the highest levels, D is shown the identity secret keys and internal state (i.e. prekey secret keys) of all parties. Honest parties are assumed to delete their one-time prekey secret keys after use, and even at $\mathsf{leak} = \mathsf{high}$ only the honest parties' final states are leaked (i.e. deleted data is excluded). Honest-but-curious accusers, however, might not make such deletions, and so at $\mathsf{disc} = \mathsf{high}$ the accusers' *initial* states are disclosed (i.e. deleted data is included).

Definition 1. *The* leakage function $\mathcal{L}_{\mathsf{leak}}$ *for the set* $\mathcal{H}$ *of accused users of a* $\mathsf{BAKE}^{\dagger}$ *protocol is defined as follows:*

$$\mathcal{L}_{\mathsf{leak}}\left((\mathsf{isk}_u, \mathsf{st}_u)_{u\in\mathcal{H}}\right) := \begin{cases} (\bot, \bot) & \text{if } \mathsf{leak} = \mathsf{low} \\ ((\mathsf{isk}_u)_{u\in\mathcal{H}}, \bot) & \text{if } \mathsf{leak} = \mathsf{med} \\ ((\mathsf{isk}_u, \mathsf{st}_u)_{u\in\mathcal{H}}) & \text{if } \mathsf{leak} = \mathsf{high} \end{cases}$$

The disclosure function $\mathcal{D}_{\mathsf{disc}}$ *for the set* $\mathcal{C}$ *of honest-but-curious accusers of a* $\mathsf{BAKE}^{\dagger}$ *protocol is defined as follows:*

$$\mathcal{D}_{\mathsf{disc}}\left((\mathsf{isk}_u, \mathsf{st}_u, \mathsf{st}_u^{\mathsf{init}})_{u\in\mathcal{C}}\right) := \begin{cases} ((\mathsf{isk}_u)_{u\in\mathcal{C}}, \bot, \bot) & \text{if } \mathsf{disc} = \mathsf{low} \\ ((\mathsf{isk}_u, \mathsf{st}_u)_{u\in\mathcal{C}}), \bot) & \text{if } \mathsf{disc} = \mathsf{med} \\ ((\mathsf{isk}_u, \mathsf{st}_u, \mathsf{st}_u^{\mathsf{init}})_{u\in\mathcal{C}}) & \text{if } \mathsf{disc} = \mathsf{high} \end{cases}$$

Since parties' internal states are shown to D, these states must be simulated in the simulated world. In addition to $\mathsf{SimTrans}$ and $\mathsf{SimPreK}$, the simulator also includes algorithms $\mathsf{SimSt}_{\mathcal{C}}$ and $\mathsf{SimSt}_{\mathcal{H}}$. The former is executed once at the end of the game to simulate all accusers' states. The latter is used to update honest receiver r's state after D directs a key exchange between paries (s, r). By necessity, $\mathsf{SimSt}_{\mathcal{H}}$ is allowed access to the secret information $(\mathsf{isk}_r, \mathsf{st}_r)$ of $r \in \mathcal{H}$, but $\mathsf{SimSt}_{\mathcal{H}}$ is "isolated" from (i.e. doesn't share state with) the rest of the simulator algorithms, and it is not given the identity s of the counterparty. Thus, if the real $\mathsf{BAKE}^{\dagger}.\mathsf{Receive}$ somehow updates st_r in a way that identifies s, this update will not be simulatable and $\mathsf{BAKE}^{\dagger}$ will not be deniable.

Last-Resort Prekeys. The deniability game enforces that each one-time prekey is used only once, see $\mathsf{counter}_r$ in Figure 5, and if one-time prekeys run out the game reverts to the use of last-resort prekeys by setting $t \leftarrow \bot$. As discussed in Section 1, many schemes are subject to a degradation in deniability guarantees for receivers who run out of one-time prekeys and are forced to use $t = \bot$.

Local and Global Deniability. Finally, the deniability game is parameterized by $\mathsf{ATK} \in \{\mathsf{local}, \mathsf{global}\}$. In the local (also called 1-out-of-2) deniability game, the distinguisher is disallowed from requesting key exchange transcripts between two honest parties $(s, r) \in \mathcal{H} \times \mathcal{H}$. Therefore, the simulator only needs to be able to simulate transcripts for key exchanges in which one of the two parties is an accuser, i.e. the simulator knows one of the two parties' secrets. In effect, then, an honest user $h \in \mathcal{H}$ in a locally deniable scheme can only deny their participation in a key exchange with counterparty c by claiming that the transcript was faked by c in an attempt to incriminate h.

Global (also called 1-out-of-infinity) deniability is a substantially stronger notion. In that game, the simulator must be able to simulate transcripts for key exchanges in which both parties are honest, i.e. the simulator doesn't know either party's secrets. (Notice, however, that distinguisher D might also not know either party's secrets, depending on parameter leak.) In a globally deniable scheme, two honest users can simultaneously deny their participation in a key exchange by claiming that the transcript was faked by any third party in an attempt to incriminate them.

Definition 2. *(Deniability Against Honest-but-Curious Accusers). Let* $\mathcal{N} := [N]$ *be the set of all users,* $\mathcal{H} \subseteq \mathcal{N}$ *be the set of accused users, and* $\mathcal{C} := \mathcal{N} \setminus \mathcal{H}$ *be the set of accusers. Protocol* $\mathsf{BAKE}^\dagger$ *is* deniable against honest-but-curious accusers *with respect to leakage* $\mathsf{leak} \in \{\mathsf{low}, \mathsf{med}, \mathsf{high}\}$*, disclosure* $\mathsf{disc} \in \{\mathsf{low}, \mathsf{med}, \mathsf{high}\}$*, KDC leakage bit* $\mathcal{L}_{\mathsf{KDC}} \in \{0, 1\}$*, and locality* $\mathsf{ATK} \in \{\mathsf{local}, \mathsf{global}\}$ *if there exists an efficient simulator* $\mathsf{Sim} = (\mathsf{SimPreK}, \mathsf{SimTrans}, \mathsf{SimSt}_{\mathcal{H}}, \mathsf{SimSt}_{\mathcal{C}})$ *such that for any efficient distinguisher* D*, for* $\mathsf{Game}^{\mathsf{ATK}}_{\mathsf{D},\mathcal{H},\mathcal{L}_{\mathsf{leak}},\mathcal{D}_{\mathsf{disc}},\mathcal{L}_{\mathsf{KDC}}}$ *specified in Fig. 4 and 5, there is a negligible function* ϵ *s.t.*

$$\Pr\left[\mathsf{Game}^{\mathsf{ATK}}_{\mathsf{D},\mathcal{H},\mathcal{L}_{\mathsf{leak}},\mathcal{D}_{\mathsf{disc}},\mathcal{L}_{\mathsf{KDC}}}(1^\kappa, \mathsf{real}) = 0\right] \leq \Pr\left[\mathsf{Game}^{\mathsf{ATK}}_{\mathsf{D},\mathcal{H},\mathcal{L}_{\mathsf{leak}},\mathcal{D}_{\mathsf{disc}},\mathcal{L}_{\mathsf{KDC}}}(1^\kappa, \mathsf{sim}) = 0\right] + \epsilon(\kappa)$$

4.2 Deniability Against Malicious Accusers

We also consider deniability against malicious accusers (also called *strong* deniability). As in Section 4.1, we use the model of [20], but adapted for the extra communication flow of $\mathsf{BAKE}^\dagger$. This strong deniability game closely resembles the honest-but-curious game above, so to avoid redundancy we defer its full specification to Appendix D and provide an overview of its important aspects here. The essential difference from the honest-but-curious deniability game is that, in the malicious deniability game, all the actions of real world accusers are determined by an accuser algorithm $\mathcal{A}$, not by honest invocations of the $\mathsf{BAKE}^\dagger$ algorithms. In particular, $\mathcal{A}$ provides all accusers' prekey bundles at the start of the game, and $\mathcal{A}$ generates the ρ, m protocol messages sent by accusers during key exchange sessions. At the end of the game, the entire state of the malicious accuser $\mathcal{A}$ is shown to distinguisher D (replacing the gradated disclosure functions of the honest-but-curious accusers above).

The simulator in the strong deniability game has a much more difficult task. It must be able to simulate the behavior and ultimate state of an arbitrary accuser

algorithm $\mathcal{A}$. In order for this to be possible, the simulator is allowed black-box oracle access to $\mathcal{A}$. Stated differently, the definition of strong deniability requires that, for any $\mathcal{A}$, there exists a simulator such that, for every D, the real and simulated worlds are not distinguished by D.

5 Integration of X3DH with PVRF-based SAS-MA

We introduce protocol X3DH+PVRF-SAS in Fig. 6. This protocol is not simply a black-box composition of X3DH and PVRF-SAS. In order to maximize the ease of integration with existing X3DH deployments, we reuse the X3DH receiver's one-time key otsk (or semi-static key ssk if one-time keys run out) as their PECVRF key. This design choice allows PVRF-SAS authentication to be added to X3DH without any change to the key distribution server. There is only a small overhead to the sender and receiver's client-side code and communication bandwidth on their first two messages exchanged. Because of this key reuse, however, security of X3DH+PVRF-SAS is not immediate. Nonetheless, Theorems 3 and 4 below state that (1) X3DH remains secure when its keys are reused for PECVRF, and (2) PECVRF remains secure when its keys are reused for X3DH. Together these theorems imply the security of X3DH+PVRF-SAS.

Theorem 3. X3DH *with* PECVRF *key reuse (Figure 6) is a key-indistinguishable bundled authenticated key exchange protocol [18, Definition 3] under the GapDH and DDH assumptions and in the ROM.*

Theorem 4. PECVRF *with* X3DH *key reuse (Figure 6) realizes* $\mathcal{F}_{\mathsf{PVRF}}$ *under the GapDH and DDH assumptions in the ROM.*

Due to space constraints we defer the proofs of Theorems 3 and 4 to Appendix E. Briefly, the former proof relies on the UC security of protocol PECVRF, but with non-black-box use of the PECVRF simulator. The latter proof is based on the observation that a DDH oracle can be used to simulate X3DH without knowledge of secret keys.

Deniability of X3DH+PVRF-SAS against passive adversaries. We specify the honest-but-curious deniability of X3DH+PVRF-SAS in the following theorem.

Theorem 5. *The* X3DH+PVRF-SAS *protocol is globally deniable with respect to disclosure function* $\mathcal{D}_{\mathsf{high}}$*, leakage function* $\mathcal{L}_{\mathsf{high}}$ *and KDC leakage bit* $\mathcal{L}_{\mathsf{KDC}} = 1$ *if the one-time prekey bundles are never depleted. If one-time prekeys run out then it is globally deniable for leakage function* $\mathcal{L}_{\mathsf{med}}$*. Specifically, if one-time prekeys run out it is still locally deniable for the sender for leakage function* $\mathcal{L}_{\mathsf{high}}$ *and it is locally deniable for the receiver and globally deniable for the sender and receiver for leakage function* $\mathcal{L}_{\mathsf{med}}$ *if* DDH *is hard and* KDF *is a random oracle.*

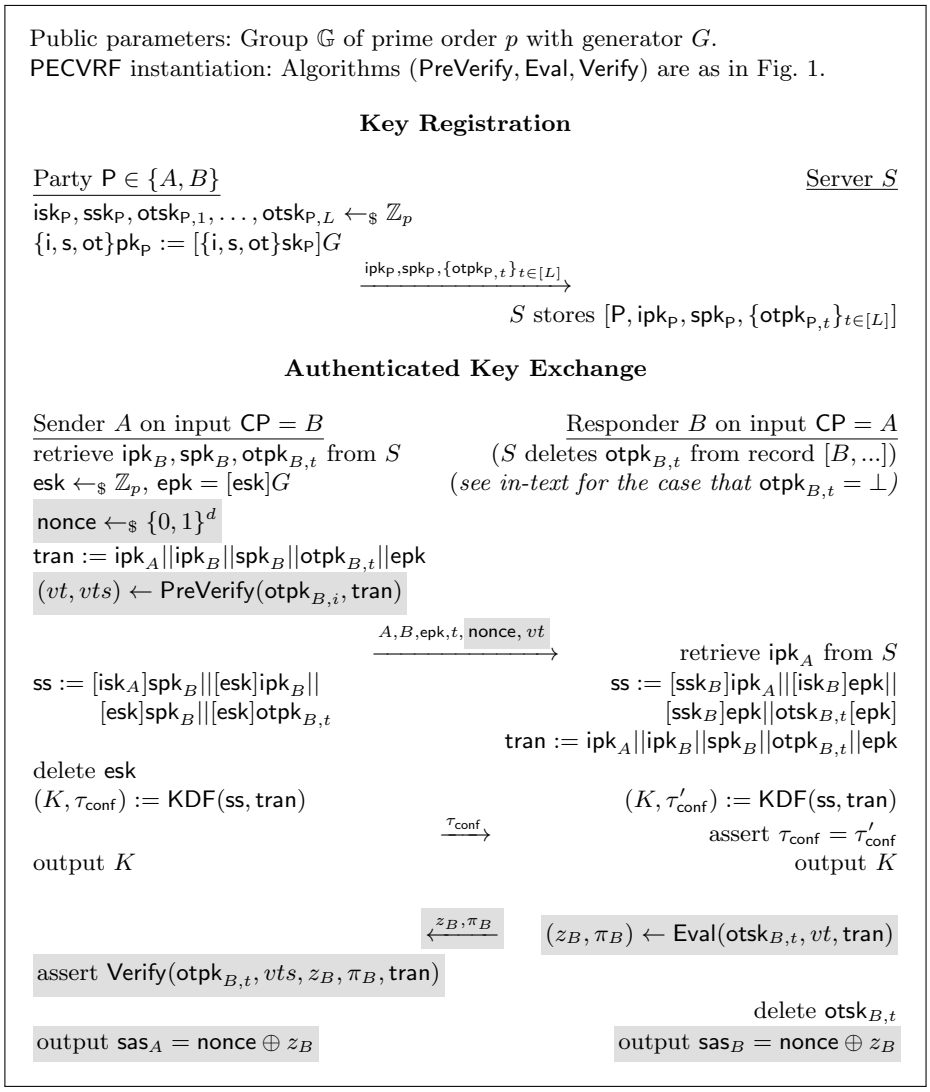

Fig. 6: Protocol X3DH+PVRF-SAS, i.e. X3DH amended by SAS-MA scheme PVRF-SAS instantiated with PECVRF (see Fig. 1) shown in gray. (Confirming that $\mathsf{sas}_A = \mathsf{sas}_B$ is left to the application level.)

Strong deniability of X3DH+PVRF-SAS. We specify the deniability of X3DH+PVRF-SAS against malicious adversaries in the following theorem:

Theorem 6. *Assuming the underlying group used in* X3DH+PVRF-SAS *is algebraic, the set of malicious accusers are algebraic algorithms over the groups used in* X3DH+PVRF-SAS *and the one-time prekeys never run out then* X3DH+PVRF-SAS *is strongly global deniable for leakage function* $\mathcal{L}_{\mathsf{high}}$ *and KDC leakage bit* $\mathcal{L}_{\mathsf{KDC}}$=1.

It is strongly local deniable for receivers and strongly global deniable for senders and receivers if $\mathsf{leak} = \mathsf{med}$ *and the last-resort prekey is used. It is strongly local deniable for senders for* $\mathsf{leak} = \mathsf{high}$ *even if last-resort prekey is used.*

We defer the details of the protocol and deniability proof to Appendix G.

References

1. C. Badertscher, P. Gazi, A. Kiayias, A. Russell, and V. Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *Computer and Communications Security - CCS*, pages 913–930, 2018.
2. C. Badertscher, P. Gaži, I. Querejeta-Azurmendi, and A. Russell. On UC-secure range extension and batch verification for ECVRF. *Cryptology ePrint Archive*, 2022. https://eprint.iacr.org/2022/1045.
3. M. Bellare and G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *Computer and Communications Security - CCS*, page 390–399, 2006.
4. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Foundations of Computer Science - FOCS*, pages 136–145, 2001.
5. K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila. A formal security analysis of the Signal messaging protocol. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 451–466, 2017.
6. R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *International Cryptology Conference - Crypto*, pages 174–187, 1994.
7. R. Cramer and V. Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *Advances in Cryptology - Eurocrypt*, pages 45–64, 2002.
8. B. David, P. Gazi, A. Kiayias, and A. Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Advances in Cryptology - Eurocrypt*, pages 66–98, 2018.
9. M. Di Raimondo, R. Gennaro, and H. Krawczyk. Deniable authentication and key exchange. In *Computer and Communications Security - CCS*, pages 400–409, 2006.
10. W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.
11. Y. Dodis, J. Katz, A. Smith, and S. Walfish. Composability and on-line deniability of authentication. In *Theory of Cryptography - TCC*, pages 146–162, 2009.
12. C. Dwork, M. Naor, and A. Sahai. Concurrent zero-knowledge. In *ACM Symposium on Theory of Computing - STOC*, pages 409–418, 1998.
13. R. Fiedler and C. Janson. A deniability analysis of signal's initial handshake pqxdh. *Cryptology ePrint Archive*, 2024. https://eprint.iacr.org/2024/741.
14. R. Fiedler and R. Langrehr. On deniable authentication against malicious verifiers. In *International Cryptology Conference - Crypto*, pages 3–38, 2025.
15. C. Gehrmann, C. J. Mitchell, and K. Nyberg. Manual authentication for wireless devices. *RSA CryptoBytes*, 7(1):24–37, 2004.
16. S. Goldberg, L. Reyzin, D. Papadopoulos, and J. Vcelak. Verifiable random functions (VRFs), internet-draft (IRTF), 2022. https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-vrf-14.

17. Y. Gu, S. Jarecki, P. Nazarian, and A. Rai. Security without trusted third parties: VRF-based authentication with short authenticated strings. In *Advances in Cryptology — Asiacrypt*, 2025.
18. K. Hashimoto, S. Katsumata, and T. Wiggers. Bundled authenticated key exchange: A concrete treatment of signal's handshake protocol and Post-Quantum security. In *34th USENIX Security Symposium*, pages 6777–6796, 2025.
19. M. Jakobsson, K. Sako, and R. Impagliazzo. Designated verifier proofs and their applications. In *Advances in Cryptology - Eurocrypt*, pages 143–154, 1996.
20. S. Katsumata, G. Niot, I. Tucker, and T. Wiggers. Comprehensive deniability analysis of signal handshake protocols: X3DH, PQXDH to fully Post-Quantum with deniable ring signatures. In *34th USENIX Security Symposium*, pages 6797–6816, 2025.
21. H. Krawczyk. HMQV: a high-performance secure Diffie-Hellman protocol. In *International Cryptology Conference - Crypto*, page 546–566, 2005.
22. S. Laur, N. Asokan, and K. Nyberg. Efficient mutual data authentication using manually authenticated strings. *Cryptology ePrint Archive*, 2005. https://eprint.iacr.org/2005/424.
23. S. Laur and K. Nyberg. Efficient mutual data authentication using manually authenticated strings. In *Conference on Cryptology and Network Security - CANS*, pages 90–107, 2006.
24. S. Laur and S. Pasini. Sas-based group authentication and key agreement protocols. In *Public Key Cryptography - PKC*, pages 197–213, 2008.
25. S. Lawlor and K. Lewi. WhatsApp key transparency. Real-World Cryptography Conference - RWC, 2024. https://iacr.org/submit/files/slides/2024/rwc/rwc2024/18/slides.pdf.
26. B. McMillion. Key transparency, internet-draft (IRTF), 2025. https://datatracker.ietf.org/doc/draft-ietf-keytrans-architecture/.
27. S. Micali, M. Rabin, and S. Vadhan. Verifiable random functions. In *Foundations of Computer Science - FOCS*, pages 120–130, 1999.
28. M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. *J. ACM*, 51(2):231–262, Mar. 2004.
29. M. Shirvanian and N. Saxena. Wiretapping via mimicry: Short voice imitation man-in-the-middle attacks on crypto phones. In *Computer and Communications Security - CCS*, page 868–879, 2014.
30. M. Stadler. Publicly verifiable secret sharing. In *Advances in Cryptology - Eurocrypt*, pages 190–199, 1996.
31. N. Unger and I. Goldberg. Deniable key exchanges for secure messaging. In *Proceedings of the 22nd acm sigsac conference on computer and communications security*, pages 1211–1223, 2015.
32. N. Unger and I. Goldberg. Improved strongly deniable authenticated key exchanges for secure messaging. *Proceedings on Privacy Enhancing Technologies*, 2018.
33. N. Vatandas, R. Gennaro, B. Ithurburn, and H. Krawczyk. On the cryptographic deniability of the signal protocol. In *International Conference on Applied Cryptography and Network Security - ACNS*, pages 188–209, 2020.
34. S. Vaudenay. Secure communications over insecure channels based on short authenticated strings. In *International Cryptology Conference - Crypto*, pages 309–326, 2005.

A Deferred Preliminaries

In this section we recall the standard cryptographic assumptions that we use.

Let $\mathbb{G}$ be a cyclic group of order p with generator g, and assume a (possibly deterministic) algorithm $\texttt{ParG}$ which generates group description $(\mathbb{G}, p, g)$ for each κ. Throughout this work we omit $\texttt{ParG}$ from our figures for brevity and leave $(\mathbb{G}, p, g)$ as an implicit input to all algorithms.

Definition 3. *The* Computational Diffie-Hellman *(CDH) assumption holds in the group family defined by* $\texttt{ParG}$ *if, for every efficient algorithm* $\mathcal{A}$*, there exists a negligible function* ϵ *s.t.*

$$\Pr_{\substack{(\mathbb{G},p,g)\leftarrow\texttt{ParG}(\kappa)\\ a,b\leftarrow_{\$}\mathbb{Z}_p}}[\mathcal{A}(p,g,g^a,g^b)\rightarrow g^{ab}]\leq\epsilon(\kappa)$$

Definition 4. *The* Decisional Diffie-Hellman *(DDH) assumption holds in the group family defined by* $\texttt{ParG}$ *if, for every efficient algorithm* $\mathcal{A}$*, there exists a negligible function* ϵ *s.t.*

$$\left|\Pr_{\substack{(\mathbb{G},p,g)\leftarrow\texttt{ParG}(\kappa)\\ a,b\leftarrow_{\$}\mathbb{Z}_p}}[\mathcal{A}(p,g,g^a,g^b,g^{ab})\rightarrow 1]-\Pr_{\substack{(\mathbb{G},p,g)\leftarrow\texttt{ParG}(\kappa)\\ a,b,c\leftarrow_{\$}\mathbb{Z}_p}}[\mathcal{A}(p,g,g^a,g^b,g^c)\rightarrow 1]\right|\leq\epsilon(\kappa)$$

Definition 5. *Define oracle* $\mathsf{DDH}_g:\mathbb{G}^3\rightarrow\{0,1\}$ *as* $\mathsf{DDH}_g(A,B,C)=1$ *if and only if there exists* $a,b\in\mathbb{Z}_p$ *s.t.* $A=g^a, B=g^b$*, and* $C=g^{ab}$*. Then, the* Gap Diffie-Hellman *(GapDH) assumption holds in the group family defined by* $\texttt{ParG}$ *if, for every efficient algorithm* $\mathcal{A}$*, there exists a negligible function* ϵ *s.t.*

$$\Pr_{\substack{(\mathbb{G},p,g)\leftarrow\texttt{ParG}(\kappa)\\ a,b\leftarrow_{\$}\mathbb{Z}_p}}[\mathcal{A}^{\mathsf{DDH}_g(\cdot,\cdot,\cdot)}(p,g,g^a,g^b)\rightarrow g^{ab}]\leq\epsilon(\kappa)$$

B PVRF Security and Deniability

In this section we include material deferred from Section 2, including the definition of UC functionality $\mathcal{F}_{\mathsf{PVRF}}$, proof that protocol PECVRF realizes $\mathcal{F}_{\mathsf{PVRF}}$, and proof of PECVRF deniability.

B.1 Security Model

In [17], the security of SAS-MA protocol $\mathsf{VRF\text{-}SAS}$ is proven when instantiated with a VRF that UC-realizes functionality $\mathcal{F}_{\mathsf{VRF}}$, as introduced by [8] and refined in [1,2]. Figure 7 presents that VRF functionality $\mathcal{F}_{\mathsf{VRF}}$ along with our new PVRF functionality $\mathcal{F}_{\mathsf{PVRF}}$. The version of $\mathcal{F}_{\mathsf{VRF}}$ shown in Figure 7 is that of [2], albeit with some reformulations in order to make $\mathcal{F}_{\mathsf{VRF}}$ and $\mathcal{F}_{\mathsf{PVRF}}$ more similar and easier to understand in relation to each other. These changes to $\mathcal{F}_{\mathsf{VRF}}$ are mostly internal, and they do not alter the security properties of the functionality at all. We explain and justify the reformulations below, after first walking through the high-level workings of the functionality.

The functionality interacts with a set of honest parties $\mathcal{P}$ and an adversary $\mathcal{A}^*$. It maintains table $F[\cdot,\cdot]$ whose entries are all initially $\perp$, and the following sets, all initially empty: S_{vk} for registered keys, $\mathcal{C}_{vk}$ for adversarial and compromised keys, S_{vt} for verification tokens, $\mathcal{C}_{vt}$ for compromised tokens and S_{pr} for proofs.

Key Generation. On $(\mathsf{KeyGen}, \mathsf{kptr})$ from any $\mathsf{P} \in \mathcal{P}$ s.t. $(\mathsf{P}, \mathsf{kptr}, \cdot) \notin S_{vk}$, hand $(\mathsf{KeyGen}, \mathsf{kptr}, \mathsf{P})$ to $\mathcal{A}^*$, and on vk from $\mathcal{A}^*$ s.t. $(\cdot, \cdot, vk) \notin S_{vk}$ do the following: $S_{vk} \leftarrow S_{vk} \cup \{(\mathsf{P}, \mathsf{kptr}, vk)\}$, return $(\mathsf{VerificationKey}, \mathsf{kptr}, \mathsf{P}, vk)$ to P

Key Compromise. On $(\mathsf{KeyComp}, \mathsf{P}, \mathsf{kptr})$ from $\mathcal{A}^*$ do the following:
if $\exists vk$ s.t. $(\mathsf{P}, \mathsf{kptr}, vk) \in S_{vk}$ then $\mathcal{C}_{vk} \leftarrow \mathcal{C}_{vk} \cup \{vk\}$

Malicious Key Generation. On $(\mathsf{MalKeyGen}, vk)$ from $\mathcal{A}^*$ s.t. $(\cdot, \cdot, vk) \notin S_{vk}$ do: $S_{vk} \leftarrow S_{vk} \cup \{(\mathcal{A}^*, \perp, vk)\}$, $\mathcal{C}_{vk} \leftarrow \mathcal{C}_{vk} \cup \{vk\}$, return $(\mathsf{VerificationKey}, vk)$ to $\mathcal{A}^*$

Pre-Verification. On $(\mathsf{PreVerify}, vk, x)$ from any $\mathsf{V} \in \mathcal{P}$, send $(\mathsf{PreVerify}, \mathsf{V}, vk, x)$ to $\mathcal{A}^*$, receive vt in response, set $S_{vt} \leftarrow S_{vt} \cup \{(\mathsf{V}, vk, x, vt)\}$, and return $(\mathsf{VerificationToken}, vt)$ to V.

Verifier Compromise. On $(\mathsf{PreVerifyComp}, \mathsf{V}, vt, vk, x)$ where $(\mathsf{V}, vk, x, vt) \in S_{vt}$: set $\mathcal{C}_{vt} \leftarrow \mathcal{C}_{vt} \cup \{(\mathsf{V}, vk, x, vt)\}$.

(P)VRF Evaluation and Proof. On $(\mathsf{Eval}, \mathsf{kptr}, x, vt)$ from P where $\exists vk$ s.t. $(\mathsf{P}, \mathsf{kptr}, vk) \in S_{vk}$, send $(\mathsf{Eval}, \mathsf{P}, \mathsf{kptr}, x, vt)$ to $\mathcal{A}^*$, and on (ϕ, π) from $\mathcal{A}^*$ do:

1. assert $\exists(\mathsf{V}, vk, x, vt) \in S_{vt}$ or $(\phi = 1)$
2. if $\exists(vk', x', vt', \pi) \in S_{pr}$, then assert $(vk', x', vt') = (vk, x, vt)$
3. if $F[vk, x] = \perp$ then set $z \leftarrow_{\$} Z$ and $F[vk, x] \leftarrow z$
4. $S_{pr} \leftarrow S_{pr} \cup \{(vk, x, vt, \pi)\}$
5. return $(\mathsf{Evaluated}, \mathsf{kptr}, x, vt, z, \pi)$ to P and to $\mathcal{A}^*$

Malicious (P)VRF Evaluation. On $(\mathsf{MalEval}, vk, x)$ from $\mathcal{A}^*$ where $vk \in \mathcal{C}_{vk}$:
1. if $F[vk, x] = \perp$ then set $z \leftarrow_{\$} Z$ and $F[vk, x] \leftarrow z$
2. return $(\mathsf{Evaluated}, vk, x, z)$ to $\mathcal{A}^*$

Verification. On $(\mathsf{Verify}, vk, x, vt, z, \pi)$ from any $\mathsf{V} \in \mathcal{P}$, send $(\mathsf{Verify}, \mathsf{V}, vk, x, vt, z, \pi)$ to $\mathcal{A}^*$, and on receiving ϕ from $\mathcal{A}^*$ do the following:

1. $f \leftarrow 1$ if $F[vk, x] = z \ \vee \ (\phi = 1 \wedge (\mathsf{V}, vk, x, vt) \in \mathcal{C}_{vt})$
   and $(vk, x, vt, \pi) \in S_{pr} \ \vee \ (\phi = 1 \wedge \neg\exists(vk', x', vt', \pi) \in S_{pr})$
   and $(\mathsf{V}, vk, x, vt) \in S_{vt} \ \vee \ (\phi = 1)$
   (otherwise $f \leftarrow 0$)
2. if $f = 1$, then $S_{pr} \leftarrow S_{pr} \cup \{(vk, x, vt, \pi)\}$
3. return $(\mathsf{Verified}, vk, x, vt, z, \pi, f)$ to V

Fig. 7: Functionalities $\mathcal{F}_{\mathsf{VRF}}$ and $\mathcal{F}_{\mathsf{PVRF}}$: ideal functionalities for VRF (omit shadowed text) and PVRF (include shadowed text) with range Z

An honest party interacting with functionality $\mathcal{F}_{\mathsf{PVRF}}$ is either a prover P or a verifier V. Prover P begins by executing KeyGen, which notifies the ideal adversary $\mathcal{A}^*$ to provide a fresh verification key vk for P. The adversary can compromise P's key using interface KeyComp and can register their own verification keys using interface MalKeyGen. Verifier V meanwhile uses interface PreVerify to create a verification token vt, which like vk is supplied by the ideal adversary. Token vt is specific to the verification key vk and function argument x on which V wishes to later verify a proof. The deniability of our PVRF protocol stems from the fact that the party who generates vt holds some secret that allows for the creation of false proofs for any statement. We therefore model the compromise of this secret via the PreVerifyComp interface, and this compromise implies that $\mathcal{A}^*$ can violate the soundness of any future verifications under vt.

P performs a function evaluation using interface Eval, and this evaluation is specific to one verification token vt under which P wishes to generate a proof. We want our model to capture PVRF schemes wherein not every vt is valid and P will simply abort if they run on an invalid vt. Therefore, we grant $\mathcal{A}^*$ the ability (via the bit ϕ) to control whether or not the prover should proceed on vt. However, $\mathcal{A}^*$ cannot claim that a vt honestly created by PreVerify is invalid. Function pseudorandomness is modeled by the fact that every function output is sampled from a random function, and $\mathcal{A}^*$ only sees these outputs for honestly-performed Eval evaluations and for adversarial MalEval evaluations, which are only allowed for compromised or adversarially-created keys.

Finally, V uses interface Verify to verify a function evaluation under a previously generated verification token vt. Verification only succeeds if π is a valid proof specifically for token vt (though $\mathcal{A}^*$ can register any fresh π as valid at the time of Verify). By default, verification fails automatically if vt was not created by V. However, $\mathcal{A}^*$ has the ability (via the bit ϕ) to override this behavior and allow verification to continue regardless. We generalize our model with this override so that $\mathcal{F}_{\mathsf{PVRF}}$ is strictly a generalization of $\mathcal{F}_{\mathsf{VRF}}$. Note that this override does not allow $\mathcal{A}^*$ to make verification succeed on statements that are false. However, if the verification token vt has been compromised, then $\mathcal{A}^*$ can actually cause successful verification of false statements, as mentioned above.

Reformulations to $\mathcal{F}_{\mathsf{VRF}}$. As mentioned above, Figure 7 presents $\mathcal{F}_{\mathsf{VRF}}$ with some reformulations relative to how it was shown in [2]. Here we list these changes and explain that they do not alter the security properties of the functionality at all. In this discussion we denote by $\mathcal{F}'_{\mathsf{VRF}}$ the unaltered functionality of [2].

1. $\mathcal{F}'_{\mathsf{VRF}}$ maintains a set S_{Eval} of (vk, x, z) tuples from every Eval execution. $\mathcal{A}^*$ can query this set from $\mathcal{F}'_{\mathsf{VRF}}$ at any time via the PastEvaluations interface, and it is also sent to $\mathcal{A}^*$ as part of the Verify message. In $\mathcal{F}_{\mathsf{VRF}}$, we remove S_{Eval} entirely and instead send every Evaluated message to $\mathcal{A}^*$ at the end of Eval.

   Any $\mathcal{A}^*$ interacting with $\mathcal{F}'_{\mathsf{VRF}}$ (and therefore expecting to see S_{Eval}) can be modified to interact with $\mathcal{F}_{\mathsf{VRF}}$, because $\mathcal{A}^*$ has enough information to simply maintain S_{Eval} internally. In particular, the Evaluated message includes

(kptr, x, z), and the mapping from kptr to vk is known by $\mathcal{A}^*$ (in fact, it is chosen by $\mathcal{A}^*$, see KeyGen).
Conversely, any $\mathcal{A}^*$ interacting with $\mathcal{F}_{\mathsf{VRF}}$ (and therefore expecting to see $\mathsf{Evaluated}$) can be modified to interact with $\mathcal{F}'_{\mathsf{VRF}}$. At the start of Eval, $\mathcal{A}^*$ is sent (kptr, x) and sends back π. Also, $\mathcal{A}^*$ knows whether the sent π will pass the assertion of not having been used for another (vk', x') statement. $\mathcal{A}^*$ can immediately call $\mathsf{PastEvaluations}$ to get S_{Eval} and learn the corresponding z, and thereby create for itself the full $\mathsf{Evaluated}$ message.

2. $\mathcal{F}'_{\mathsf{VRF}}$ allows $\mathcal{A}^*$ to perform $\mathsf{MalEval}$ on $vk \notin \mathcal{C}$ as long as an honest party has already evaluated $F[vk, x]$. $\mathcal{F}_{\mathsf{VRF}}$ removes this option, but $\mathcal{A}^*$ can easily simulate $\mathsf{MalEval}$ calls on $vk \notin \mathcal{C}$ using the $\mathsf{Evaluated}$ messages previously received.
3. $\mathcal{F}'_{\mathsf{VRF}}$ maintains information about previously seen proofs using two tables T and P. The former maps each (vk, x) pair to a set of valid π's. The latter maps each valid π to the corresponding (vk, x). $\mathcal{F}_{\mathsf{VRF}}$ unifies this bookkeeping to set S_{pr} of (vk, x, π) tuples. This change is seen in Eval, $\mathsf{MalEval}$, and Verify, but it is only internal to the functionality.
4. In $\mathcal{F}'_{\mathsf{VRF}}$, Verify checks that $(\cdot, \cdot, vk) \in S_{pk}$ and $F[vk, x] = z$, but the former condition is implied by the latter, so $\mathcal{F}_{\mathsf{VRF}}$ omits it.
5. $\mathcal{F}'_{\mathsf{VRF}}$ allows $\mathcal{A}^*$ to call Verify. $\mathcal{F}_{\mathsf{VRF}}$ removes this option, but $\mathcal{A}^*$ can easily simulate Verify calls internally. As previously noted, $\mathcal{A}^*$ already knows the entire table F of function values, and $\mathcal{A}^*$ can also keep track of S_{pr}, which is only updated in Eval and Verify with values provided by $\mathcal{A}^*$. Thus, $\mathcal{A}^*$ can predict the result f of any Verify query. In $\mathcal{F}'_{\mathsf{VRF}}$, $\mathcal{A}^*$ can also use Verify to add proofs to S_{pr}, but $\mathcal{A}^*$ can also simulate this power with $\mathcal{F}_{\mathsf{VRF}}$ by simply remembering what proofs should have been added and adding them later when first verified by an honest party.

Relation Between $\mathcal{F}_{\mathsf{VRF}}$ and $\mathcal{F}_{\mathsf{PVRF}}$. As mentioned in Section 2, $\mathcal{F}_{\mathsf{PVRF}}$ does not capture the privacy/deniability property of PVRFs. As a result, functionality $\mathcal{F}_{\mathsf{PVRF}}$ is a generalization of (i.e. strictly weaker than) $\mathcal{F}_{\mathsf{VRF}}$. In other words, if protocol Π realizes $\mathcal{F}_{\mathsf{VRF}}$ then Π also realizes $\mathcal{F}_{\mathsf{PVRF}}$ (with the minor change of adding empty strings as vt wherever called for by the $\mathcal{F}_{\mathsf{PVRF}}$ syntax). To show this, we describe how a Π simulator Sim in interaction with $\mathcal{F}_{\mathsf{VRF}}$ can be compiled into a simulator that instead interacts with $\mathcal{F}_{\mathsf{PVRF}}$.

1. On $(\mathsf{PreVerify}, \mathsf{V}, vk, x)$ from $\mathcal{F}_{\mathsf{PVRF}}$, respond with an empty string as vt.
2. On $(\mathsf{Eval}, \mathsf{P}, \mathsf{kptr}, x, vt)$ from $\mathcal{F}_{\mathsf{PVRF}}$, forward that message (omitting vt) to Sim, and receive response π. Send $(1, \pi)$ to $\mathcal{F}_{\mathsf{PVRF}}$. (In effect, allow Eval to proceed regardless of vt.)
3. On $(\mathsf{Verify}, \mathsf{V}, vk, x, vt, z, \pi)$ from $\mathcal{F}_{\mathsf{PVRF}}$, forward that message (omitting V and vt) to Sim, and receive response ϕ. Set $\phi' \leftarrow F[vk, x] = z \wedge ((vk, x, vt, \pi) \in S_{pr} \vee (\phi = 1 \wedge \neg\, \exists (vk', x', vt', \pi) \in S_{pr}))$, and send it to $\mathcal{F}_{\mathsf{PVRF}}$.

B.2 Proof of Security

In this section we prove that protocol PECVRF (Figure 1) realizes $\mathcal{F}_{\mathsf{PVRF}}$.

Theorem 7. PECVRF *realizes* $\mathcal{F}_{\mathsf{PVRF}}$*, assuming that* H_i*,* H_o*,* H_a*,* H_b*, and* H_{FS} *are modeled as random oracles and that the* DDH *problem is hard in* $\mathbb{G}$*.*

Specifically, for any efficient adversary $\mathcal{A}^*$ against protocol PECVRF, there exists a simulator SIM such that no efficient environment $\mathcal{Z}$ can distinguish the view of $\mathcal{A}^*$ interacting with the real PECVRF protocol and the view of SIM interacting with the ideal functionality $\mathcal{F}_{\mathsf{PVRF}}$ with advantage better than $2 \cdot \mathsf{Adv}_{\mathsf{DDH}} + q_{\mathsf{H}_i} \cdot q_{\mathsf{H}_o} \cdot \mathsf{Adv}_{\mathsf{CDH}} + \frac{q_{\mathsf{KeyGen}} \cdot (q_{\mathsf{KeyGen}} + q_{\mathsf{H}_i}) + (q_{\mathsf{PreVerify}} + 1) \cdot (q_{\mathsf{PreVerify}} + q_{\mathsf{H}_{\mathsf{FS}}}) + q_{\mathsf{H}_i} \cdot (q_{\mathsf{H}_i} + q_{\mathsf{H}_o})}{p}$ where there are at most q_{KeyGen} executions of KeyGen, there are at most $q_{\mathsf{PreVerify}}$ executions of PreVerify, there are at most q_{H_i} queries to H_i, H_a, and H_b (including those that occur in PreVerify, Eval, and Verify), there are at most q_{H_o} queries to H_o, $\mathsf{Adv}_{\mathsf{DDH}}$ is a bound on the DDH advantage of any efficient algorithm, and $\mathsf{Adv}_{\mathsf{CDH}}$ is a bound on the probability that any efficient algorithm solves CDH.

Proof. For any adversary $\mathcal{A}^*$, we construct simulator $\mathsf{SIM}_{\mathsf{PECVRF}}$ as shown in Figure 8. Without loss of generality, we assume that $\mathcal{A}^*$ is a "dummy" adversary that merely passes messages to and from the environment $\mathcal{Z}$.

We now show that, for any efficient (i.e. PPT) $\mathcal{Z}$, the distinguishing advantage of $\mathcal{Z}$ between the real and simulated worlds is negligible. We denote the probability that $\mathcal{Z}$ outputs 1 in the interaction with the real world game or the simulated game as resp. $\mathbf{Real}_{\mathcal{Z}}[\mathsf{PECVRF}]$ and $\mathbf{Ideal}_{\mathcal{Z}}[\mathcal{F}_{\mathsf{PVRF}}, \mathsf{SIM}_{\mathsf{PECVRF}}]$. We denote the probability that $\mathcal{Z}$ outputs 1 in an interaction with security game G_i as $\mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_i] = \Pr[1 \leftarrow \mathcal{Z}^{\mathsf{G}_i}(1^\kappa)]$, and we define $\Delta(\mathsf{G}_i, \mathsf{G}_j) = |\mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_i] - \mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_j]|$. The argument proceeds by a series of game changes, starting with game G_0 s.t. $\mathsf{G}_0 = \mathbf{Real}_{\mathcal{Z}}[\mathsf{PECVRF}]$ and ending with game G_8 s.t. $\mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_8] = \mathbf{Ideal}_{\mathcal{Z}}[\mathcal{F}_{\mathsf{PVRF}}, \mathsf{SIM}_{\mathsf{PECVRF}}]$. We must show that $\Delta(\mathsf{G}_0, \mathsf{G}_8)$ is negligible.

Game G_0 (the real world): The distinguisher $\mathcal{Z}$ interacts with PECVRF (Figure 1) in the role of the honest parties and in the role of the adversary. H_i, H_o, H_a, H_b, and H_{FS} are true random oracles.

$$\mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_0] = \mathbf{Real}_{\mathcal{Z}}[\mathsf{PECVRF}]$$

Game G_1 (KeyGen never collides): G_1 is G_0 with only one change: after sampling a random vk in KeyGen, abort if vk was previously sampled by another KeyGen instance or if a $\mathsf{H}_i(\cdot, vk)$, $\mathsf{H}_a(\cdot, vk)$, or $\mathsf{H}_b(\cdot, vk)$ query was previously made.

The probability of a collision is bounded by a simple birthday bound. If no collision occurs, G_0 and G_1 are identical.

$$\Delta(\mathsf{G}_0, \mathsf{G}_1) \leq \frac{q_{\mathsf{KeyGen}} \cdot (q_{\mathsf{KeyGen}} + q_{\mathsf{H}_i})}{p}$$

Game G_2 (H_i, H_a, and H_b outputs are correlated): G_2 is G_1 with the following changes:

– Maintain two tables $T[\cdot, \cdot]$ and $R[\cdot, \cdot]$ (initially all empty).

Initialize empty tables $T[\cdot,\cdot]$, $R[\cdot,\cdot]$ and empty sets S_{vk}, $\mathcal{C}_{vk}$, S_{vt}, S_{eval}.

Sample $T[vk,x], R[vk,x] \leftarrow_\$ \mathbb{Z}_p$ whenever first referenced, and:

- abort if the same $T[vk,x]$ was previously sampled for a different (vk',x')
- abort if there was a previous query $\mathsf{H}_o(w)$ s.t. $w = vk^{T[vk,x]}$

On $(\mathsf{KeyGen}, \mathsf{kptr}, \mathsf{P})$ from $\mathcal{F}_{\mathsf{PVRF}}$:

- sample $k \leftarrow_\$ \mathbb{Z}_p$ and set $vk := g^k$
- abort if $\exists(\cdot,\cdot,vk) \in S_{vk}$ or there was previous query $(vk,\cdot)$ to H_i, H_a, or H_b
- set $S_{vk} \leftarrow S_{vk} \cup \{(\mathsf{kptr}, k, vk)\}$ and return vk to $\mathcal{F}_{\mathsf{PVRF}}$

On $(\mathsf{PreVerify}, \mathsf{V}, vk, x)$ from $\mathcal{F}_{\mathsf{PVRF}}$:

- sample $\alpha, \beta \leftarrow_\$ \mathbb{Z}_p$, and set $h := g^\alpha \cdot \mathsf{H}_i(vk,x)^\beta$ and $h' := \mathsf{H}_a(vk,x)^\alpha \cdot \mathsf{H}_b(vk,x)^\beta$
- sample $c, s_1, s_2 \leftarrow_\$ \mathbb{Z}_p$ and set $\tau = (c,(s_1,s_2))$
- compute $\eta := g^{s_1} \cdot \mathsf{H}_i(vk,x)^{s_2} \cdot h^c$ and $\eta' := \mathsf{H}_a(vk,x)^{s_1} \cdot \mathsf{H}_b(vk,x)^{s_2} \cdot h'^c$
- program $\mathsf{H}_{\mathsf{FS}}(vk,x,h,h',\eta,\eta') := c$
- set $vt := (h,h',\tau)$ and $S_{vt} \leftarrow S_{vt} \cup \{(\mathsf{V},vk,x,vt,\alpha,\beta)\}$, and return vt to $\mathcal{F}_{\mathsf{PVRF}}$

On $(\mathsf{Eval}, \mathsf{P}, \mathsf{kptr}, x, vt)$ from $\mathcal{F}_{\mathsf{PVRF}}$:

- parse $(h,h',\tau) \leftarrow vt$ and $(c,(s_1,s_2)) \leftarrow \tau$
- set $\eta := g^{s_1} \cdot \mathsf{H}_i(vk,x)^{s_2} \cdot h^c$ and $\eta' := \mathsf{H}_a(vk,x,h)^{s_1} \cdot \mathsf{H}_b(vk,x,h)^{s_2} \cdot h'^c$, and if $c \neq \mathsf{H}_{\mathsf{FS}}(vk,x,h,h',\eta,\eta')$, then return $(\phi,\pi) := (0,\perp)$ to $\mathcal{F}_{\mathsf{PVRF}}$
- otherwise, set $w := vk^{T[vk,x]}$ and $v := (h')^{1/R[vk,x]}$, and return $(\phi,\pi) := (1,(w,v))$ to $\mathcal{F}_{\mathsf{PVRF}}$

On $(\mathsf{Evaluated}, \mathsf{kptr}, x, vt, z, \pi)$ from $\mathcal{F}_{\mathsf{PVRF}}$:

- retrieve $(\mathsf{kptr}, k, vk) \in S_{vk}$ and set $S_{eval} \leftarrow S_{eval} \cup \{(vk,x,z)\}$

On $(\mathsf{Verify}, \mathsf{V}, vk, x, vt, z, \pi)$ from $\mathcal{F}_{\mathsf{PVRF}}$:

- if not $\exists(\mathsf{V},vk,x,vt,\cdot,\cdot) \in S_{vt}$, then return $\phi := 0$ to $\mathcal{F}_{\mathsf{PVRF}}$
- parse $(w,v) \leftarrow \pi$, and set $f := 1$ if $z = \mathsf{H}_o(w)$ and $w = vk^{T[vk,x]}$; otherwise, set $f := 0$
- return $\phi := f$ to $\mathcal{F}_{\mathsf{PVRF}}$

On fresh query (vk,x) to H_i, H_a, or H_b, program:

- $\mathsf{H}_i(vk,x) := g^{T[vk,x]}$, $\mathsf{H}_a(vk,x) := vk^{R[vk,x]}$, and $\mathsf{H}_b(vk,x) := vk^{T[vk,x] \cdot R[vk,x]}$

On fresh query $\mathsf{H}_o(w)$:

- find (vk,x) s.t. $w = vk^{T[vk,x]}$ (if it exists in T)
- if no such (vk,x) exists, then program $\mathsf{H}_o(w) \leftarrow_\$ Z$ and return
- if $\exists(\cdot,\cdot,vk) \in S_{vk}$, then retrieve $(vk,x,z) \in S_{eval}$ (abort if it doesn't exist), program $\mathsf{H}_o(w) := z$, and return
- if $vk \notin \mathcal{C}_{vk}$, then send $(\mathsf{MalKeyGen}, vk)$ to $\mathcal{F}_{\mathsf{PVRF}}$ and set $\mathcal{C}_{vk} \leftarrow \mathcal{C}_{vk} \cup \{vk\}$
- send $(\mathsf{MalEval}, vk, x)$ to $\mathcal{F}_{\mathsf{PVRF}}$, receive z in response, and program $\mathsf{H}_o(w) := z$

Program H_{FS} at random, except when programmed by $\mathsf{PreVerify}$ (see above)

On compromise of party P's record $(\mathsf{kptr}, -, vk)$:

- send $(\mathsf{KeyComp}, \mathsf{P}, \mathsf{kptr})$ to $\mathcal{F}_{\mathsf{PVRF}}$, retrieve $(\mathsf{kptr}, k, vk) \in S_{vk}$, and send (kptr, k, vk) to $\mathcal{A}^*$

On compromise of party V's record $(vt, vk, x, -, -)$:

- send $(\mathsf{PreVerifyComp}, \mathsf{V}, vt, vk, x)$ to $\mathcal{F}_{\mathsf{PVRF}}$, retrieve $(\mathsf{V}, vk, x, vt, \alpha, \beta) \in S_{vt}$, and send $(vt, vk, x, \alpha, \beta)$ to $\mathcal{A}^*$

Fig. 8: $\mathsf{SIM}_{\mathsf{PECVRF}}$: Simulator for protocol PECVRF

– On fresh query $\mathsf{H}_i(vk, x)$, $\mathsf{H}_a(vk, x)$, or $\mathsf{H}_b(vk, x)$, sample $T[vk, x], R[vk, x] \leftarrow_\$ \mathbb{Z}_p$. Program $\mathsf{H}_i(vk, x) := g^{T[vk,x]}$ and $\mathsf{H}_a(vk, x) := vk^{R[vk,x]}$. If vk was previously sampled and output by KeyGen, then program $\mathsf{H}_b(vk, x) := vk^{R[vk,x]\cdot T[vk,x]}$; otherwise, sample random $\mathsf{H}_b(vk, x) \leftarrow_\$ \mathbb{G}$ as before.

Given an environment $\mathcal{Z}$ that can distinguish G_1 from G_2, we construct a reduction $\mathcal{R}$ that can solve DDH. The input to $\mathcal{R}$ is tuple $(A, B, C) \in \mathbb{G}^3$. $\mathcal{R}$ emulates game G_1 to $\mathcal{Z}$ with the following modification. On fresh query $\mathsf{H}_i(vk, x)$, $\mathsf{H}_a(vk, x)$, or $\mathsf{H}_b(vk, x)$ where vk was previously sampled and output by KeyGen, do:

– Sample $\delta, \mu, \nu \leftarrow_\$ \mathbb{Z}_p$.
– Retrieve k s.t. $vk = g^k$.
– Program $\mathsf{H}_i(vk, x) := A^\delta$, $\mathsf{H}_a(vk, x) := (g^\mu \cdot B^\nu)^k$, and $\mathsf{H}_b(vk, x) := (A^\mu \cdot C^\nu)^{\delta\cdot k}$.

Here we apply the well-known DDH randomized self-reduction [30,28]. If (A, B, C) is a Diffie-Hellman tuple, then $(A^\delta, g^\mu \cdot B^\nu, (A^\mu \cdot C^\nu)^\delta)$ is a random Diffie-Hellman tuple. If not, then it is a uniformly random tuple. If (A, B, C) is not a Diffie-Hellman tuple, then the random oracles are programmed independently at random, so the view of $\mathcal{Z}$ is identical to that in G_1. If (A, B, C) is a Diffie-Hellman tuple, then $(\mathsf{H}_i, \mathsf{H}_a, \mathsf{H}_b)$ is a random Diffie-Hellman tuple for every (vk, x), so the view of $\mathcal{Z}$ is identical to that in G_2.

Reduction $\mathcal{R}$ outputs 1 iff $\mathcal{Z}$ outputs 1. Thus, the advantage of $\mathcal{Z}$ at distinguishing between G_1 and G_2 is equal to the advantage of $\mathcal{R}$ in the DDH game, which is $\leq \mathsf{Adv}_{\mathsf{DDH}}$.

$$\Delta(\mathsf{G}_1, \mathsf{G}_2) \leq \mathsf{Adv}_{\mathsf{DDH}}$$

Game G_3 (Eval uses trapdoors): G_3 is G_2 with the following changes to Eval after verifying vt:

– Sample $T[vk, x], R[vk, x] \leftarrow_\$ \mathbb{Z}_p$ if they are undefined.
– Instead of computing $w := \mathsf{H}_i(vk, x)^k$, compute $w := vk^{T[vk,x]}$.
– Instead of computing $v := h^k$, compute $v := (h')^{1/R[vk,x]}$.

In G_3, Eval uses trapdoors instead of secret keys to compute its responses. Due to the programming of H_i, w is unconditionally identical between G_2 and G_3. If there exists $\alpha, \beta \in \mathbb{Z}_p$ s.t. $h = g^\alpha \cdot \mathsf{H}_i(vk, x)^\beta$ and $h' = \mathsf{H}_a(vk, x)^\alpha \cdot \mathsf{H}_b(vk, x)^\beta$ (as NIZK τ is meant to ensure), then $(h')^{1/R[vk,x]} = (\mathsf{H}_a(vk, x)^\alpha \cdot \mathsf{H}_b(vk, x)^\beta)^{1/R[vk,x]} = vk^\alpha \cdot vk^{T[vk,x]\cdot\beta} = (g^\alpha \cdot \mathsf{H}_i(vk, x)^\beta)^k = h^k$, i.e. v is identical between G_2 and G_3. Thus, G_2 and G_3 are identical unless there is a break in the soundness of a vt NIZK τ.

Consider algorithm $\mathcal{B}$, which on input $c_1, \ldots, c_{q_{\mathsf{H}_{\mathsf{FS}}}} \in \mathbb{Z}_p$ executes $\mathcal{Z}^{\mathsf{G}_3}$, programming each H_{FS} query using its inputs in succession. $\mathcal{B}$ monitors every $vt = (h, h', \tau)$ submitted by $\mathcal{Z}$ to Eval. $\mathcal{B}$ parses and verifies NIZK $\tau = (c, (s_1, s_2))$ by computing $\eta := g^{s_1} \cdot \mathsf{H}_i(vk, x)^{s_2} \cdot h^c$ and $\eta' := \mathsf{H}_a(vk, x)^{s_1} \cdot \mathsf{H}_b(vk, x)^{s_2} \cdot h'^c$ and then checking that $c = \mathsf{H}_{\mathsf{FS}}(vk, x, h, h', \eta, \eta')$. If verification passes, then

$\mathcal{B}$ checks whether there exists $\alpha, \beta \in \mathbb{Z}_p$ s.t. $h = g^\alpha \cdot \mathsf{H}_i(vk, x)^\beta$ and $h' = \mathsf{H}_a(vk, x)^\alpha \cdot \mathsf{H}_b(vk, x)^\beta$. If not (i.e. soundness is broken), then $\mathcal{B}$ immediately halts and outputs (j, τ), where $(vk, x, h, h', \eta, \eta')$ was the jth query to H_{FS} (and therefore $c = c_j$). If $\mathcal{Z}^{\mathsf{G}_3}$ ends without triggering this event, then $\mathcal{B}$ halts in a failure.[5]

When run on random $c_1, \ldots, c_{q_{\mathsf{H}_{\mathsf{FS}}}} \leftarrow_\$ \mathbb{Z}_p$, $\mathcal{B}$ succeeds iff there is a break in the soundness of a vt NIZK τ in game G_3. We can use the forking lemma [3] to upper-bound the probability $acc_{\mathcal{B}}$ of this event. In particular, the forking lemma compiles $\mathcal{B}$ into an algorithm $\mathcal{F}(\mathcal{B})$ that either fails or outputs (τ, τ^*) where $\tau = (c, (s_1, s_2))$ and $\tau^* = (c^*, (s_1^*, s_2^*))$ are verifying NIZKs output by two different executions of $\mathcal{B}$ with the same $(vk, x, h, h', \eta, \eta')$ but different $c \neq c^*$. If $\mathcal{F}(\mathcal{B})$ succeeds, then we can extract witness (α, β) from (τ, τ^*). In particular, $\alpha = (s_1 - s_1^*)/(c^* - c)$ and $\beta = (s_2 - s_2^*)/(c^* - c)$. However, $\mathcal{B}$ is defined to only output NIZKs for which no such α, β exists. Thus, the success of $\mathcal{F}(\mathcal{B})$ is a contradiction, and its probability $acc_{\mathcal{F}(\mathcal{B})} = 0$. The forking lemma therefore implies that $acc_{\mathcal{B}} \leq q_{\mathsf{H}_{\mathsf{FS}}}/p$.

$$\Delta(\mathsf{G}_2, \mathsf{G}_3) \leq \frac{q_{\mathsf{H}_{\mathsf{FS}}}}{p}$$

Game G_4 (honest function outputs are unpredictable): G_4 is G_3 with only one change: abort if the environment ever queries $\mathsf{H}_o(vk^{T[vk,x]})$ for any vk, x s.t. some honest party P performed $(\mathsf{KeyGen}, \mathsf{kptr})$ that outputted vk and P has not performed $(\mathsf{Eval}, \mathsf{kptr}, x, vt)$ (that didn't fail due to invalid vt). Clearly G_3 and G_4 are identical unless this abort occurs (denote the event of this abort as E).

To upper-bound the probability of E in G_4, we first define a modified game G_4':

- At the start of execution, sample $i \leftarrow_\$ [q_{\mathsf{H}_i}]$.
- For the ith $(\hat{vk}, \hat{x})$ queried to H_i, H_a, H_b, or Eval s.t. $\hat{vk}$ was previously sampled by some honest party P performing $(\mathsf{KeyGen}, \hat{\mathsf{kptr}})$, program $\mathsf{H}_i(\hat{vk}, \hat{x})$, $\mathsf{H}_a(\hat{vk}, \hat{x}), \mathsf{H}_b(\hat{vk}, \hat{x}) \leftarrow_\$ \mathbb{G}$ (rather than sampling and using trapdoors T, R as in G_4).
- If, after $(\mathsf{KeyGen}, \hat{\mathsf{kptr}})$, P performs $(\mathsf{Eval}, \hat{\mathsf{kptr}}, \hat{x}, vt)$, then compute $w := \mathsf{H}_i(\hat{vk}, \hat{x})^{\hat{k}}$ and $v := h^{\hat{k}}$ (rather than using the T, R trapdoors as in G_4).
- Trigger abort E if the environment ever queries $\mathsf{H}_o(\mathsf{H}_i(\hat{vk}, \hat{x})^{\hat{k}})$ and P has not performed $(\mathsf{Eval}, \hat{\mathsf{kptr}}, \hat{x}, vt)$ (that didn't fail due to invalid vt).

We construct a DDH reduction $\mathcal{R}_{\mathsf{DDH}}$ to show that E occurs with near-equal probability in G_4 and G_4'. The input to $\mathcal{R}_{\mathsf{DDH}}$ is tuple $(A, B, C) \in \mathbb{G}^3$. $\mathcal{R}_{\mathsf{DDH}}$ emulates game G_4' to $\mathcal{Z}$, except that it programs $\mathsf{H}_i(\hat{vk}, \hat{x}) := A$, $\mathsf{H}_a(\hat{vk}, \hat{x}) := B$, and $\mathsf{H}_b(\hat{vk}, \hat{x}) := C$. If (A, B, C) is a Diffie-Hellman tuple, then the view of $\mathcal{Z}$ is identical to that in G_4. If (A, B, C) is not a Diffie-Hellman tuple, then the view

[5] $\mathcal{B}$ is not an efficient algorithm, but this is not of importance to our argument. For intuition, recall that the soundness of a Σ protocol is statistical, not computational.

of $\mathcal{Z}$ is identical to that in G'_4. Reduction $\mathcal{R}_{\mathsf{DDH}}$ outputs 1 iff event E occurs. Thus, $|\Pr[E \leftarrow \mathcal{Z}^{\mathsf{G}_4}] - \Pr[E \leftarrow \mathcal{Z}^{\mathsf{G}'_4}]| \leq \mathsf{Adv}_{\mathsf{DDH}}$.

Given an environment $\mathcal{Z}$ that can trigger E in G'_4, we construct a reduction $\mathcal{R}_{\mathsf{CDH}}$ that can solve CDH. The input to $\mathcal{R}_{\mathsf{CDH}}$ is $(A, B) \in \mathbb{G}^2$. $\mathcal{R}_{\mathsf{CDH}}$ emulates game G'_4 with the following modifications:

- At the start of execution, after sampling $i \leftarrow_\$ [q_{\mathsf{H}_i}]$, also sample $j \leftarrow_\$ [q_{\mathsf{H}_o}]$.
- If, after $(\mathsf{KeyGen}, \hat{\mathsf{kptr}})$, P performs $(\mathsf{Eval}, \hat{\mathsf{kptr}}, \hat{x}, vt)$ (that doesn't fail due to invalid vt), then immediately halt in failure.
- In every execution of KeyGen, sample $s \leftarrow_\$ \mathbb{Z}_p$ and set $vk := A^s$. (Notice that, in G'_4, secret key k is not used.) Denote by $\hat{s}$ the value sampled by the KeyGen instance that outputs $\hat{vk}$.
- Program $\mathsf{H}_i(\hat{vk}, \hat{x}) := B$.
- Upon the jth query $\mathsf{H}_o(w)$ to H_o, immediately output $w^{1/\hat{s}}$ and halt.

If event E occurs in game G'_4 on indices i, j (meaning that the jth query to H_o is $\mathsf{H}_o(\mathsf{H}_i(\hat{vk}, \hat{x})^{\hat{k}})$), then the view of $\mathcal{Z}$ in $\mathcal{R}_{\mathsf{CDH}}$ is identical to that in G'_4 and $\mathcal{R}_{\mathsf{CDH}}$ wins CDH. Thus, $\Pr[E \leftarrow \mathcal{Z}^{\mathsf{G}'_4}]/(q_{\mathsf{H}_i} \cdot q_{\mathsf{H}_o}) \leq \mathsf{Adv}_{\mathsf{CDH}}$. Combining our results, we reach a bound on the probability of E in G_4.

$$\Delta(\mathsf{G}_3, \mathsf{G}_4) \leq \mathsf{Adv}_{\mathsf{DDH}} + q_{\mathsf{H}_i} \cdot q_{\mathsf{H}_o} \cdot \mathsf{Adv}_{\mathsf{CDH}}$$

Game G_5 ($\mathsf{PreVerify}$ NIZKs are simulated): G_5 is G_4 with the following changes to how τ is computed in $\mathsf{PreVerify}$ (after computing h, h' as before):

- Sample $c, s_1, s_2 \leftarrow_\$ \mathbb{Z}_p$ and set $\tau = (c, (s_1, s_2))$.
- Compute $\eta := g^{s_1} \cdot \mathsf{H}_i(vk, x)^{s_2} \cdot h^c$ and $\eta' := \mathsf{H}_a(vk, x)^{s_1} \cdot \mathsf{H}_b(vk, x)^{s_2} \cdot h'^c$.
- Program $\mathsf{H}_{\mathsf{FS}}(vk, x, h, h', \eta, \eta') := c$.

G_4 and G_5 are identically distributed as long as this NIZK simulation succeeds. However, it can fail if $\mathsf{H}_{\mathsf{FS}}(vk, x, h, h', \eta, \eta')$ was already queried by the environment or programmed by a previous instance of $\mathsf{PreVerify}$. Since the input is randomly sampled by $\mathsf{PreVerify}$, the probability of this event is bounded by a simple birthday bound.

$$\Delta(\mathsf{G}_4, \mathsf{G}_5) \leq \frac{q_{\mathsf{PreVerify}} \cdot (q_{\mathsf{PreVerify}} + q_{\mathsf{H}_{\mathsf{FS}}})}{p}$$

Game G_6 (Verify is perfectly sound): G_6 is G_5 with the following changes to how f is computed in $(\mathsf{Verify}, vk, x, vt, z, \pi)$:

- Parse $(w, v) \leftarrow \pi$.
- If $z \neq \mathsf{H}_o(w)$ then return $f := 0$.
- Otherwise, do not check $v = vk^\alpha \cdot w^\beta$ as before. Instead, set $f := 1$ iff $w = vk^{T[vk,x]}$.

G_5 and G_6 are identical unless $\mathcal{Z}$ is able to violate the soundness of π. In particular, define E as the event that (for some instance of Verify) $v = vk^{\alpha} \cdot w^{\beta}$ but there does not exist k s.t. $vk = g^k$ and $w = \mathsf{H}_i(vk, x)^k$. Since $\mathsf{H}_i(vk, x) = g^{T[vk,x]}$, the latter condition is equivalently written as $w \neq vk^{T[vk,x]}$. The security of SPHFs ensures that E is negligibly likely.

Consider algorithm $\mathcal{B}$, which on input $\beta_1, \ldots, \beta_{q_{\mathsf{PreVerify}}} \in \mathbb{Z}_p$ executes $\mathcal{Z}^{\mathsf{G}_6}$ with the following modifications:

- In the jth instance of $\mathsf{PreVerify}$, instead of sampling $\alpha, \beta \leftarrow_{\$} \mathbb{Z}_p$, sample $\gamma \leftarrow_{\$} \mathbb{Z}_p$ and set $h := g^{\gamma}$ and $h' := \mathsf{H}_a(vk, x)^{\gamma}$. Define (and store) $\alpha := \gamma - T[vk, x] \cdot \beta_j$ and $\beta := \beta_j$.
- If event E occurs, immediately halt and output (j, β_j), where the relevant vt was produced by the jth instance of $\mathsf{PreVerify}$.
- If $\mathcal{Z}^{\mathsf{G}_6}$ ends without triggering E, then $\mathcal{B}$ halts in a failure.

When run on random $\beta_1, \ldots, \beta_{q_{\mathsf{PreVerify}}} \leftarrow_{\$} \mathbb{Z}_p$, the view of $\mathcal{Z}$ is identical in G_6 and in $\mathcal{B}$ (recall from game G_2 that $(\mathsf{H}_i, \mathsf{H}_a, \mathsf{H}_o)$ are a Diffie-Hellman tuple for every vk, x). $\mathcal{B}$ succeeds iff event E occurs in game G_6. We can use the forking lemma [3] to upper-bound the probability $acc_{\mathcal{B}}$ of this event. In particular, the forking lemma compiles $\mathcal{B}$ into an algorithm $\mathcal{F}(\mathcal{B})$ that either fails or outputs (β, β^*) for verifying SPHFs output by two different executions of $\mathcal{B}$ with the same (vk, x, γ, w, v) but different $\beta \neq \beta^*$ (because changing $\beta_1, \ldots, \beta_{q_{\mathsf{PreVerify}}}$ across two executions of $\mathcal{B}$ does not alter the view of $\mathcal{Z}$ as long as E does not occur).

The success of $\mathcal{F}(\mathcal{B})$ implies that $v = vk^{\gamma - T[vk,x] \cdot \beta} \cdot w^{\beta} = vk^{\gamma - T[vk,x] \cdot \beta^*} \cdot w^{\beta^*}$. Rearranging terms, we get $w^{\beta^* - \beta} = vk^{T[vk,x] \cdot (\beta^* - \beta)}$ and (since $\beta \neq \beta^*$) $w = vk^{T[vk,x]}$. However, $\mathcal{B}$ is defined to only succeed if $w \neq vk^{T[vk,x]}$. Thus, the success of $\mathcal{F}(\mathcal{B})$ is a contradiction, and its probability $acc_{\mathcal{F}(\mathcal{B})} = 0$. The forking lemma therefore implies that $acc_{\mathcal{B}} \leq q_{\mathsf{PreVerify}}/p$.

$$\Delta(\mathsf{G}_5, \mathsf{G}_6) \leq \frac{q_{\mathsf{PreVerify}}}{p}$$

Game G_7 (H_o queries are linked to H_i queries): G_7 is G_6 with the following changes:

- After sampling $T[vk, x]$, abort if the same value was previously sampled for a different (vk', x').
- After sampling $T[vk, x]$, abort if there was a previous query $\mathsf{H}_o(w)$ s.t. $w = vk^{T[vk,x]}$.
- On fresh query $\mathsf{H}_o(w)$, find (vk, x) s.t. $w = vk^{T[vk,x]}$ (if it exists in T). In this game, nothing is done with (vk, x) after it is found.

This game change uses the trapdoors in $\mathsf{H}_i(vk, x)$ outputs to link $\mathsf{H}_o(w)$ queries to their corresponding (vk, x), if it exists. The two aborts guarantee that this linkage is (1) unique and (2) matches are not retroactively created by coincidence.

The probability of an abort is bounded by a simple birthday bound. If no abort occurs, G_6 and G_7 are identical.

$$\Delta(\mathsf{G}_6, \mathsf{G}_7) \leq \frac{q_{\mathsf{H}_i} \cdot (q_{\mathsf{H}_i} + q_{\mathsf{H}_o})}{p}$$

Game G_8 (the simulated world): The change from G_7 to G_8 is purely conceptual. All interactions between the game and honest parties now match the code of $\mathcal{F}_{\mathsf{PVRF}}$, so one can imagine the monolithic simulator now cleanly split into two components $\mathcal{F}_{\mathsf{PVRF}}$ and $\mathsf{SIM}_{\mathsf{PECVRF}}$.

$$\Delta(\mathsf{G}_7, \mathsf{G}_8) = 0$$
$$\mathsf{Exp}_{\mathcal{Z}}[\mathsf{G}_8] = \mathbf{Ideal}_{\mathcal{Z}}[\mathcal{F}_{\mathsf{PVRF}}, \mathsf{SIM}_{\mathsf{PECVRF}}]$$

Summing up the distinguishing advantage bounds for each incremental game change yields an overall bound on $\mathcal{Z}$'s distinguishing advantage between the real and simulated worlds.

$$\Delta(\mathsf{G}_0, \mathsf{G}_8) \leq 2 \cdot \mathsf{Adv}_{\mathsf{DDH}} + q_{\mathsf{H}_i} \cdot q_{\mathsf{H}_o} \cdot \mathsf{Adv}_{\mathsf{CDH}} + \frac{q_{\mathsf{KeyGen}} \cdot (q_{\mathsf{KeyGen}} + q_{\mathsf{H}_i}) + (q_{\mathsf{PreVerify}} + 1) \cdot (q_{\mathsf{PreVerify}} + q_{\mathsf{H}_{\mathsf{FS}}}) + q_{\mathsf{H}_i} \cdot (q_{\mathsf{H}_i} + q_{\mathsf{H}_o})}{p}$$

$p = |\mathbb{G}|$ is an exponential function of the security parameter. If $\mathcal{Z}$ is efficient, then q_{KeyGen}, $q_{\mathsf{PreVerify}}$, q_{H_i}, q_{H_o}, and $q_{\mathsf{H}_{\mathsf{FS}}}$ are polynomial functions of the security parameter. Furthermore, $\mathsf{Adv}_{\mathsf{DDH}}$ and $\mathsf{Adv}_{\mathsf{CDH}}$ are negligible functions of the security parameter under the DDH assumption. Thus, the distinguishing advantage of any efficient $\mathcal{Z}$ between the real and simulated worlds is negligible.

Protocol PECVRF realizes $\mathcal{F}_{\mathsf{PVRF}}$.

B.3 Proof of Deniability

In this section we prove Theorems 1 and 2, i.e. standard and strong deniability of protocol PECVRF.

Proof (Theorem 1).

We define an algorithm $\mathsf{FakeEval}$ which on input (vk, vt, vts, x) creates an indistinguishable (z, π). Parse $(\alpha, \beta) \leftarrow vts$ and sample $w \leftarrow_{\$} \mathbb{G}$ and sets $v = vk^{\alpha} \cdot w^{\beta}$. Output $(\mathsf{H}_o(w), (w, v))$. Since this is how v is constructed in the algorithm for regular verification when in possession of the verification token secret this v trivially passes the verification algorithm.

Let Game_0 be the game when $b = 0$ in $\mathsf{Game}^{\mathsf{Den}}_{\mathsf{D,PVRF}}$ and Game_1 be the case when $b = 1$. Consider a reduction to DDH which on challenge (A, B, C) sets vk as A. When $\mathcal{A}$ queries H_i on input (vk, x) (or after $\mathcal{A}$ outputs x without querying H_i), sample random field elements $\mu, \nu \xleftarrow{\$} \mathbb{Z}_p$ and program the random oracle at $\mathsf{H}_i(A, x) \leftarrow g^{\mu} \cdot B^{\nu}$. Then, set $w = A^{\mu} \cdot C^{\nu}$ and $v = A^{\alpha} \cdot (A^{\mu} \cdot C^{\nu})^{\beta}$. Finally, construct z and π as in Fig. 1.

Here we apply the well-known DDH randomized self-reduction [30,28]. If (A, B, C) is a Diffie-Hellman tuple, then $(A \cdot g^{\mu} \cdot B^{\nu}, A^{\mu} \cdot C^{\nu})$ is a random Diffie-Hellman tuple starting with A, and our reduction yields w and v as if $b = 0$ and $\mathsf{sk} = \log_g(A)$. If not, then it is a uniformly random tuple starting with A, and our reduction yields w and v that are distributed identically to the w and v computed by $\mathsf{FakeEval}$ when $b = 1$.

$$\mathsf{Adv}^{\mathsf{Den}}_{\mathsf{D,PECVRF}} \leq \mathsf{Adv}_{\mathsf{DDH}}$$

Proof (Theorem 2). We define an algorithm $\mathsf{FakeEval}$ which on interaction with any algebraic $\mathcal{A}$ creates an indistinguishable transcript compared to a real execution of Eval unless $\mathcal{A}$ can break DDH in $\mathbb{G}$. $\mathsf{FakeEval}$ runs $\mathcal{A}$ internally and retrieves outputs $vt = (h, h', \tau)$ and message x. Since $\mathcal{A}$ is algebraic they must also output a vector of field elements which defines the output group element as a combination of all the group elements seen thus far. Thus, on top of verifying the regular Fiat-Shamir NIZK using τ, $\mathsf{FakeEval}$ can directly check whether $h = g^{\alpha} \cdot \mathsf{H}_i(vk, x)^{\beta}$ and $h' = \mathsf{H}_a(vk, x)^{\alpha} \cdot \mathsf{H}_b(vk, x)^{\beta}$ for the same α and β. If so, it samples $w \leftarrow_{\$} \mathbb{G}$ and sets $v = vk^{\alpha} \cdot w^{\beta}$. Output $(\mathsf{H}_o(w), (w, v))$. Soundness of the NIZK proof of existence of α and β for h and h' ensures that an honest person would abort in the case where $\mathsf{FakeEval}_{\mathcal{A}}$ cannot extract correct α and β from the vector output by the algebraic adversary except with negligible probability. In this case $\mathsf{FakeEval}$ also aborts and does not construct a response matching the real transcript. Since this is how v is constructed in the algorithm for regular verification when in possession of the verification token secret this v trivially passes the regular verification check.

Using the same DDH randomized self-reduction as Theorem 1 we get,

$$\mathsf{Adv}^{\mathsf{SDen}}_{\mathsf{D},\mathcal{A},\mathsf{PECVRF}} \leq \mathsf{Adv}_{\mathsf{DDH}}$$

C PVRF-SAS Security

In this section we include material deferred from Section 3, including the definition of SAS-MA security, and the proof that protocol PVRF-SAS is a secure SAS-MA scheme.

C.1 Security Model

We model SAS-MA security of scheme Π via a security game $\mathsf{Game}^{\mathsf{SASma}}_{\mathcal{A},\Pi}$ shown in Figure 9. This security game is a variant of SAS-MA security defined in [17] for *groups*, specialized to the case of 2 parties and asymmetric 2-flow protocols where only responder party P_2 holds a long-term reusable key. Since the model involves only two parties we omit adaptive compromise of SAS channels, but we expand the attack conditions by allowing compromise of the secrets used by either party on the attacked session, as long as the compromise occurs after the attacked session terminates. In more details, the security game allows adversary $\mathcal{A}$ to trigger parties $\mathsf{P}_1, \mathsf{P}_2$ to execute the SAS-MA protocol repeatedly, each time on arbitrary inputs m_1, m_2 except that in each instance P_2 must be invoked on a *fresh* message m_2.[6] On any protocol instance $\mathcal{A}$ can deliver arbitrary protocol messages to parties, but it cannot change the SAS values exchanged in that instance. The adversary $\mathcal{A}$ *wins* the game if it fixes some protocol instance before its execution and on that instance it holds that $m_1 \neq m_2$ but $\mathsf{sas}_1 = \mathsf{sas}_2$.

[6] An honest party must assure this e.g. by prepending the message with a nonce. In our application in Section 5, m is fresh because it is a key exchange transcript.

$\underline{\mathsf{Game}^{\mathsf{SASma}}_{\mathcal{A},\Pi}(\kappa)}$:

1. for $i = 1, 2$ and all $\mathtt{sid}$, set $\mathsf{phase}_i^{\mathtt{sid}}$ to 0 and $\mathsf{m}_i^{\mathtt{sid}}, \mathsf{state}_i^{\mathtt{sid}}, \mathsf{sas}_i^{\mathtt{sid}}$ to $\perp$
2. set ATK to $\perp$, $\mathsf{used}, \mathsf{comp}_1$ to $\emptyset$, and comp_2 to F
3. set $(k, vk) \leftarrow \mathsf{PreProc}(1^\kappa)$
4. $\mathtt{sid} \leftarrow \mathcal{A}^{\Pi_1, \Pi_2, \Pi_3, \mathsf{FixTarget}, \mathsf{Comp}_1, \mathsf{Comp}_2}(vk)$
5. assert $\mathtt{sid} = \mathsf{ATK} \wedge \mathsf{phase}_1^{\mathtt{sid}} = 2 \wedge \mathsf{phase}_2^{\mathtt{sid}} = 1$
6. if $\mathsf{m}_1^{\mathtt{sid}} \neq \mathsf{m}_2^{\mathtt{sid}} \ \wedge \ \mathsf{sas}_1^{\mathtt{sid}} = \mathsf{sas}_2^{\mathtt{sid}}$ then return 1 else return 0

$\underline{\Pi_1(\mathtt{sid}, vk^*, m)}$:
assert $\mathsf{phase}_1^{\mathtt{sid}} = 0$
$(pm_1, \mathsf{st}_1^{\mathtt{sid}}) \leftarrow \mathsf{Init}(vk^*, m)$
$\mathsf{phase}_1^{\mathtt{sid}} \leftarrow 1$, $\mathsf{m}_1^{\mathtt{sid}} \leftarrow m$
return pm_1

$\underline{\Pi_2(\mathtt{sid}, m, pm_1^*)}$:
assert $\mathsf{phase}_2^{\mathtt{sid}} = 0$ and $m \notin \mathsf{used}$
$(pm_2, \mathsf{sas}_2^{\mathtt{sid}}) \leftarrow \mathsf{Resp}(k, m, pm_1^*)$
$\mathsf{phase}_2^{\mathtt{sid}} \leftarrow 1$, $\mathsf{m}_2^{\mathtt{sid}} \leftarrow m$
$\mathsf{used} \leftarrow \mathsf{used} \cup \{m\}$
return $(pm_2, \mathsf{sas}_2^{\mathtt{sid}})$

$\underline{\mathsf{Comp}_2()}$:
assert $(\mathsf{ATK} \neq \perp \wedge \mathsf{sas}_1^{\mathsf{ATK}} \neq \perp \wedge \mathsf{sas}_2^{\mathsf{ATK}} \neq \perp)$
$\mathsf{comp}_2 \leftarrow \mathsf{T}$, return k

$\underline{\Pi_3(\mathtt{sid}, pm_2^*)}$:
assert $\mathsf{phase}_1^{\mathtt{sid}} = 1$
$\mathsf{sas}_2^{\mathtt{sid}} \leftarrow \mathsf{InitFin}(\mathsf{st}_1^{\mathtt{sid}}, pm_2^*)$
$\mathsf{phase}_2^{\mathtt{sid}} \leftarrow 2$
return $\mathsf{sas}_2^{\mathtt{sid}}$

$\underline{\mathsf{FixTarget}(\mathtt{sid})}$:
assert $\mathsf{ATK} = \perp$
assert $\mathsf{phase}_1^{\mathtt{sid}} = \mathsf{phase}_2^{\mathtt{sid}} = 0$
$\mathsf{ATK} \leftarrow \mathtt{sid}$

$\underline{\mathsf{Comp}_1(\mathtt{sid})}$:
assert $\mathsf{phase}_1^{\mathtt{sid}} \geq 1$
assert $(\mathsf{ATK} \neq \mathtt{sid} \vee (\mathsf{sas}_1^{\mathsf{ATK}} \neq \perp \wedge \mathsf{sas}_2^{\mathsf{ATK}} \neq \perp)$
$\mathsf{comp}_1 \leftarrow \mathsf{comp}_1 \cup \{\mathtt{sid}\}$
return $\mathsf{st}_1^{\mathtt{sid}}$

Fig. 9: Security game for SAS-MMA scheme $\Pi = (\mathsf{PreProc}, \mathsf{Init}, \mathsf{Resp}, \mathsf{InitFin})$

Game $\mathsf{Game}^{\mathsf{SASma}}_{\mathcal{A},\Pi}$ models adversary $\mathcal{A}$'s interaction with honest parties via three oracles, Π_1, Π_2, Π_3, which model resp. P_1's execution of Init, P_2's execution of Resp, and P_1's execution of $\mathsf{InitFin}$, and the game uses set used to ensure that each instance of P_2 runs on a *fresh* input m. The adversary can adaptively compromise P_2's key vk (and learn the corresponding secret key k) via oracle call Comp_2, and it can adaptively compromise P_1's state $\mathsf{st}_1^{\mathtt{sid}}$ for any protocol instance $\mathtt{sid}$ via oracle call Comp_1. To win the game, however, $\mathcal{A}$ must use oracle $\mathsf{FixTarget}$ to commit to some protocol instance $\mathtt{sid}$ it hopes to attack, and (1) $\mathcal{A}$ cannot choose an instance that either party has already begun, (2) $\mathcal{A}$ cannot compromise P_2 before choosing and completing the attacked instance, and (3) $\mathcal{A}$ cannot compromise P_1 on the attacked instance until it is complete.

Definition 6. *A SAS-MA scheme Π is* secure *if for all efficient algorithms $\mathcal{A}$ there exists a negligible function ϵ s.t.* $\Pr[1 \leftarrow \mathsf{Game}^{\mathsf{SASma}}_{\mathcal{A},\Pi}(1^\kappa)] \leq 1/2^d + \epsilon(\kappa)$.

C.2 Proof of Security

In this section we prove that protocol PVRF-SAS is a secure SAS-MA scheme.

Theorem 8. *If* PVRF *realizes functionality* $\mathcal{F}_{\mathsf{PVRF}}$ *(Figure 7), then protocol* PVRF-SAS *(Figure 3) is a secure SAS-MA scheme (Definition 6).*

Proof. We argue that SAS-MA scheme PVRF-SAS is *secure* through a sequence of transformations. Throughout the transformations we fix the adversarial algorithm $\mathcal{A}_0$, and we start from G_0, see Figure 10, which is identical to the SAS-MA security experiment $\mathsf{Game}^{\mathsf{SASma}}_{\mathcal{A}_0,\Pi}(1^\kappa)$ for $\Pi = \mathsf{PVRF\text{-}SAS}$. Let $\epsilon_0 = \Pr[1 \leftarrow \mathsf{G}_0]$. Since G_0 is identical to the SAS-MA security experiment for scheme PVRF-SAS, our goal is to show that $\epsilon_0 \leq 1/2^d + \epsilon(\kappa)$ where ϵ is negligible.

1. for $i = 1, 2$ and all $\mathtt{sid}$, set $\mathsf{phase}_i^{\mathtt{sid}}$ to 0 and $\mathsf{m}_i^{\mathtt{sid}}, \mathsf{state}_i^{\mathtt{sid}}, \mathsf{sas}_i^{\mathtt{sid}}$ to $\perp$
2. set ATK to $\perp$, $\mathsf{used}, \mathsf{comp}_1$ to $\emptyset$, and comp_2 to F
3. set $(k, vk) \leftarrow \mathsf{KeyGen}(1^\kappa)$
4. $\mathtt{sid} \leftarrow \mathcal{A}^{\Pi_1,\Pi_2,\Pi_3,\mathsf{FixTarget},\mathsf{Comp}_1,\mathsf{Comp}_2}(vk)$
5. assert $\mathtt{sid} = \mathsf{ATK} \wedge \mathsf{phase}_1^{\mathtt{sid}} = 2 \wedge \mathsf{phase}_2^{\mathtt{sid}} = 1$
6. if $\mathsf{m}_1^{\mathtt{sid}} \neq \mathsf{m}_2^{\mathtt{sid}} \ \wedge \ \mathsf{sas}_1^{\mathtt{sid}} = \mathsf{sas}_2^{\mathtt{sid}}$ then return 1 else return 0

$\underline{\Pi_1(\mathtt{sid}, vk^*, m)}$:
assert $\mathsf{phase}_1^{\mathtt{sid}} = 0$
$(pm_1, \mathsf{st}_1^{\mathtt{sid}}) \leftarrow \mathsf{Init}(vk^*, m)$
$\mathsf{nonce} \leftarrow_{\$} \{0,1\}^d$
$(vt, vts) \leftarrow \mathsf{PreVerify}(vk^*, m)$
$\mathsf{st}_1^{\mathtt{sid}} \leftarrow (vk^*, m, \mathsf{nonce}, vts)$
$\mathsf{phase}_1^{\mathtt{sid}} \leftarrow 1$, $\mathsf{m}_1^{\mathtt{sid}} \leftarrow m$
return (vt, nonce)

$\underline{\Pi_2(\mathtt{sid}, m, pm_1^*)}$:
assert $\mathsf{phase}_2^{\mathtt{sid}} = 0$ and $m \notin \mathsf{used}$
$(vt^*, \mathsf{nonce}^*) \leftarrow pm_1^*$
$(z, \pi) \leftarrow \mathsf{Eval}(k, vt^*, m)$
$\mathsf{phase}_2^{\mathtt{sid}} \leftarrow 1$, $\mathsf{m}_2^{\mathtt{sid}} \leftarrow m$
$\mathsf{used} \leftarrow \mathsf{used} \cup \{m\}$
return $((z, \pi), z \oplus \mathsf{nonce}^*)$

$\underline{\mathsf{Comp}_2()}$:
assert $(\mathsf{ATK} \neq \perp \wedge \mathsf{sas}_1^{\mathsf{ATK}} \neq \perp \wedge \mathsf{sas}_2^{\mathsf{ATK}} \neq \perp)$
$\mathsf{comp}_2 \leftarrow \mathsf{T}$, return k

$\underline{\Pi_3(\mathtt{sid}, pm_2^*)}$:
assert $\mathsf{phase}_1^{\mathtt{sid}} = 1$
$(vk^*, m, \mathsf{nonce}, vts) \leftarrow \mathsf{st}_1^{\mathtt{sid}}$
$(z^*, \pi^*) \leftarrow pm_2^*$
assert $\mathsf{Verify}(vk^*, vts, m, z^*, \pi^*)$
$\mathsf{phase}_2^{\mathtt{sid}} \leftarrow 2$
return $z^* \oplus \mathsf{nonce}$

$\underline{\mathsf{FixTarget}(\mathtt{sid})}$:
assert $\mathsf{ATK} = \perp$
assert $\mathsf{phase}_1^{\mathtt{sid}} = \mathsf{phase}_2^{\mathtt{sid}} = 0$
$\mathsf{ATK} \leftarrow \mathtt{sid}$

$\underline{\mathsf{Comp}_1(\mathtt{sid})}$:
assert $\mathsf{phase}_1^{\mathtt{sid}} \geq 1$
assert $(\mathsf{ATK} \neq \mathtt{sid} \vee (\mathsf{sas}_1^{\mathsf{ATK}} \neq \perp \wedge \mathsf{sas}_2^{\mathsf{ATK}} \neq \perp)$
$\mathsf{comp}_1 \leftarrow \mathsf{comp}_1 \cup \{\mathtt{sid}\}$
return $\mathsf{st}_1^{\mathtt{sid}} = (vk^*, m, \mathsf{nonce}, vts)$

Fig. 10: Game G_0: The security game $\mathsf{Game}^{\mathsf{SASma}}_{\mathcal{A},\Pi}$, Fig. 9, instantiated with the SAS-MA scheme PVRF-SAS of Fig. 3. In yellow we show the code specific to scheme PVRF-SAS and in blue we highlight PVRF calls within that code.

Game G_0 (real world): G_0 is the security experiment $\mathsf{Game}^{\mathsf{SASma}}_{\mathcal{A}_0,\Pi}(1^\kappa)$, see Fig. 9, for Π instantiated as PVRF-SAS. We present this game explicitly in Figure 10, using standard font for the code that comes from the generic security game $\mathsf{Game}^{\mathsf{SASma}}_{\mathcal{A}_0,\Pi}$ and using yellow and blue to mark the code that pertains to the specific SAS-MA scheme PVRF-SAS, where the latter color is used whenever PVRF-SAS's code contains calls to the PVRF scheme PVRF.

1. for $i = 1, 2$ and all $\mathtt{sid}$, set $\mathsf{phase}_i^{\mathtt{sid}}$ to 0 and $\mathsf{m}_i^{\mathtt{sid}}, \mathsf{state}_i^{\mathtt{sid}}, \mathsf{sas}_i^{\mathtt{sid}}$ to $\perp$
2. set ATK, all $F[\cdot,\cdot]$ to $\perp$, $\mathsf{used}, \mathsf{comp}_1$, S_{pr} to $\emptyset$, and comp_2 to F
3. $vk \leftarrow \mathcal{A}(1^\kappa)$
4. $\mathtt{sid} \leftarrow \mathcal{A}^{\Pi_1,\Pi_2,\Pi_3,\mathsf{FixTarget},\mathsf{Comp}_1,\mathsf{Comp}_2,\mathsf{MalEval}}(vk)$
5. assert $\mathtt{sid} = \mathsf{ATK} \wedge \mathsf{phase}_1^{\mathtt{sid}} = 2 \wedge \mathsf{phase}_2^{\mathtt{sid}} = 1$
6. if $\mathsf{m}_1^{\mathtt{sid}} \neq \mathsf{m}_2^{\mathtt{sid}} \ \wedge \ \mathsf{sas}_1^{\mathtt{sid}} = \mathsf{sas}_2^{\mathtt{sid}}$ then return 1 else return 0

$\underline{\Pi_1(\mathtt{sid}, vk^*, m, vt):}$
assert $\mathsf{phase}_1^{\mathtt{sid}} = 0$
$(pm_1, \mathsf{st}_1^{\mathtt{sid}}) \leftarrow \mathsf{Init}(vk^*, m)$
$\mathsf{nonce} \leftarrow_\$ \{0,1\}^d$
$\mathsf{st}_1^{\mathtt{sid}} \leftarrow (vk^*, m, \mathsf{nonce}, vt)$
$\mathsf{phase}_1^{\mathtt{sid}} \leftarrow 1$, $\mathsf{m}_1^{\mathtt{sid}} \leftarrow m$
return (vt, nonce)

$\underline{\Pi_2(\mathtt{sid}, m, pm_1^*, \pi):}$
assert $\mathsf{phase}_2^{\mathtt{sid}} = 0$ and $m \notin \mathsf{used}$
$(vt^*, \mathsf{nonce}^*) \leftarrow pm_1^*$
if $\exists (vk', m', vt', \pi) \in S_{pr}$:
  assert $(vk', m', vt') = (vk, m, vt^*)$
if $F[vk, m] = \perp$: $F[vk, m] \leftarrow_\$ \{0,1\}^d$
$z \leftarrow F[vk, m]$
$S_{pr} \leftarrow S_{pr} \cup \{(vk, m, vt^*, \pi)\}$
$\mathsf{phase}_2^{\mathtt{sid}} \leftarrow 1$, $\mathsf{m}_2^{\mathtt{sid}} \leftarrow m$
$\mathsf{used} \leftarrow \mathsf{used} \cup \{m\}$
return $((z, \pi), z \oplus \mathsf{nonce}^*)$

$\underline{\mathsf{Comp}_2():}$
assert $(\mathsf{ATK} \neq \perp \wedge \mathsf{sas}_1^{\mathsf{ATK}} \neq \perp \wedge \mathsf{sas}_2^{\mathsf{ATK}} \neq \perp)$
$\mathsf{comp}_2 \leftarrow \mathsf{T}$

$\underline{\Pi_3(\mathtt{sid}, pm_2^*):}$
assert $\mathsf{phase}_1^{\mathtt{sid}} = 1$
$(vk^*, m, \mathsf{nonce}, vt) \leftarrow \mathsf{st}_1^{\mathtt{sid}}$
$(z^*, \pi^*) \leftarrow pm_2^*$
assert $F[vk^*, m] = z^*$ or $\mathtt{sid} \in \mathsf{comp}_1$
if $\exists (vk', m', vt', \pi^*) \in S_{pr}$:
  assert $(vk', m', vt') = (vk^*, m, vt)$
$S_{pr} \leftarrow S_{pr} \cup \{(vk^*, m, vt, \pi^*)\}$
$\mathsf{phase}_2^{\mathtt{sid}} \leftarrow 2$
return $z^* \oplus \mathsf{nonce}$

$\underline{\mathsf{FixTarget}(\mathtt{sid}):}$
assert $\mathsf{ATK} = \perp$
assert $\mathsf{phase}_1^{\mathtt{sid}} = \mathsf{phase}_2^{\mathtt{sid}} = 0$
$\mathsf{ATK} \leftarrow \mathtt{sid}$

$\underline{\mathsf{Comp}_1(\mathtt{sid}):}$
assert $\mathsf{phase}_1^{\mathtt{sid}} \geq 1$
assert $(\mathsf{ATK} \neq \mathtt{sid} \vee (\mathsf{sas}_1^{\mathsf{ATK}} \neq \perp \wedge \mathsf{sas}_2^{\mathsf{ATK}} \neq \perp)$
$\mathsf{comp}_1 \leftarrow \mathsf{comp}_1 \cup \{\mathtt{sid}\}$

$\underline{\mathsf{MalEval}(vk^*, m):}$
assert $(vk^* \neq vk \vee \mathsf{comp}_2 = \mathsf{T})$
if $F[vk^*, m] = \perp$: $F[vk^*, m] \leftarrow_\$ \{0,1\}^d$
return $F[vk^*, m]$

Fig. 11: Game G_1: Replacing PVRF calls from G_0 with $\mathcal{F}_{\mathsf{PVRF}}$ code.

Game G_1 (implanting $\mathcal{F}_{\mathsf{PVRF}}$): In G_1 we replace the real-world PVRF scheme acting in G_0 with the ideal-world functionality $\mathcal{F}_{\mathsf{PVRF}}$. In Figure 11, which shows G_1, all the calls to the PVRF scheme made by G_0, denoted in blue in Fig. 10, are replaced by the code of the ideal functionality $\mathcal{F}_{\mathsf{PVRF}}$ (Figure 7), denoted in green in Fig. 11. Since the adversary in the ideal-world (of PVRF) has additional interfaces, namely malicious key generation and PVRF evaluation via MalKeyGen and MalEval, we add these interfaces in G_1. Note that other than replacing the blue-colored code fragments in G_0 with the green-colored fragments in G_1, the remaining code of the two games is identical.

Since adversarial interfaces change between G_0 and G_1, the second game involves a formally different adversarial algorithm, denoted $\mathcal{A}_1$. Indeed, the transformation from G_0 to G_1 is justified by the assumption that scheme PVRF realizes the UC PVRF functionality $\mathcal{F}_{\mathsf{PVRF}}$. Technically, what this fact implies is that for every efficient environment algorithm $\mathcal{Z}$ and adversary $\mathcal{A}_0$, there exists an efficient algorithm *simulator* Sim s.t. $\mathcal{Z}$'s view of an interaction with $\mathcal{A}_0$ and honest parties running scheme PVRF, is computationally indistinguishable from $\mathcal{Z}$'s view of an interaction with the simulator Sim and honest parties who interact with the ideal PVRF functionality $\mathcal{F}_{\mathsf{PVRF}}$. Define $\mathcal{Z}$ as an algorithm which runs all the code of G_0 shown in Fig. 10, including adversary $\mathcal{A}_0$ as a subroutine run by $\mathcal{Z}$, except for the blue-colored fragments which correspond to honest parties' commands to PVRF. Using this notation, G_0 is an interaction between $\mathcal{Z}$ using $\mathcal{A}_0$ as a subroutine, with honest $\mathsf{P}_1, \mathsf{P}_2$ running scheme PVRF. By the assumption that PVRF realizes $\mathcal{F}_{\mathsf{PVRF}}$, there exists a simulator Sim s.t. this interaction is indistinguishable from an interaction of the same $\mathcal{Z}$ with Sim and honest $\mathsf{P}_1, \mathsf{P}_2$ interacting with $\mathcal{F}_{\mathsf{PVRF}}$. The transformation between G_0 and G_1 is an instance of this: (1) the adversary $\mathcal{A}_1$ used as subroutine in G_1 is a composition of adversary $\mathcal{A}_0$ and simulator Sim, (2) the green-colored fragments of G_1 correspond to $\mathcal{F}_{\mathsf{PVRF}}$ processing $\mathsf{P}_1, \mathsf{P}_2$ calls and $\mathcal{A}_1$'s direct calls (via oracles MalKeyGen, MalEval, and KeyComp), and (3) the remaining code of G_1 is the environment $\mathcal{Z}$. By the assumption that PVRF securely realizes $\mathcal{F}_{\mathsf{PVRF}}$, it follows that there is a negligible function $\varepsilon_{\mathsf{PVRF}}$ s.t. if $\epsilon_1 = \Pr[1 \leftarrow \mathsf{G}_1]$ then

$$\epsilon_0 \leq \epsilon_1 + \varepsilon_{\mathsf{PVRF}}(\kappa)$$

Note that $\mathcal{A}_1$'s interfaces in G_1 are different than $\mathcal{A}_0$'s interfaces to these oracles in G_0. In the real world, honest party P_2 does not need any additional inputs to run PVRF.KeyGen, but in the ideal world when P_1 generates a key via the KeyGen interface of $\mathcal{F}_{\mathsf{PVRF}}$ it should supply a key identifier kptr. Further, $\mathcal{F}_{\mathsf{PVRF}}$ reacts to (KeyGen, kptr) by sending the message (KeyGen, kptr, P_1) to adversary $\mathcal{A}_1$, who plays the role of the ideal world adversary $\mathcal{A}^*$ of Fig. 7, asking $\mathcal{A}_1$ to supply a fresh public key vk. In G_1 we short-cut this interaction by asking $\mathcal{A}_1$ to supply vk at the game's start. There is only one invocation of KeyGen in G_1, so we simply omit the identifier kptr from the game.

$\mathcal{F}_{\mathsf{PVRF}}$ on command (PreVerify, vk, x) from V notifies the ideal adversary who then supplies the verification token vt. We short-cut this interaction in G_1 by simply having $\mathcal{A}_1$ supply vt as part of their Π_1 oracle query.

Similarly, on command $(\mathsf{Eval}, \mathsf{kptr}, x, vt)$ from P, $\mathcal{F}_{\mathsf{PVRF}}$ notifies the ideal adversary who then supplies the proof π. This interaction is also short-cutted in G_1 by simply including π as part of $\mathcal{A}_1$'s Π_2 oracle query. The ideal adversary can also cause Eval to abort using bit $\phi = 0$ (if vt^* is not honestly generated), but we omit this interaction since $\mathcal{A}_1$ can equivalently just skip such calls to Π_2.

For the same reason as in Π_2's Eval call, we also assume that $\phi = 1$ in Π_3's Verify call. Also, we merge $\mathcal{F}_{\mathsf{PVRF}}$'s $\mathsf{KeyComp}$ (resp. $\mathsf{PreVerifyComp}$) bookkeeping mechanism $\mathcal{C}_{vk}$ (resp. $\mathcal{C}_{vt}$) with our game's Comp_2 (resp. Comp_1) bookkeeping mechanism comp_2 (resp. comp_1). Since vk^*, m, and nonce are already public to $\mathcal{A}_1$ before querying Comp_1, we remove them from the output of that oracle. We make one final short-cut by observing that a $(\mathsf{MalKeyGen}, vk^*)$ command has no effect unless followed by at least one $(\mathsf{MalEval}, vk^*, \cdot)$ evaluation. We therefore omit $\mathsf{MalKeyGen}$ with the implicit understanding that $\mathcal{A}_1$ would invoke it prior to any corresponding $\mathsf{MalEval}$ call.

Game G_2 (eliminating proofs): In G_2 we simplify G_1 based on the observation that the adversary can always predict whether their G_1 oracle calls will violate an "assert" statement and cause an abort, and that the adversary gains nothing by doing so. Therefore, G_2 removes the checks for proof validity from Π_2 and instead simply assumes that correct z values and proofs have been forwarded from Π_1. Proofs and verification tokens consequently play no role in G_2, which allows them to be removed from Π_1 as well. We define a new adversary $\mathcal{A}_2$ who runs $\mathcal{A}_1$ as a subroutine, internally performs the assert checks that have been removed from G_2, and only forwards those Π_1 and Π_2 calls that would not have violated those checks (with π and vt excised from the calls in order to match the G_2 interface). $\mathcal{A}_2$ succeeds in G_2 iff $\mathcal{A}_1$ succeeds in G_1, i.e. $\epsilon_1 = \epsilon_2$, where we define $\epsilon_2 = \Pr[1 \leftarrow \mathsf{G}_2]$.

It remains to upper-bound the adversary's probability of success in G_2. For the attacked session $\mathsf{ATK} = \mathtt{sid}$, consider the relevant oracle calls $\Pi_1(\mathtt{sid}, vk^*, m)$, $\Pi_2(\mathtt{sid}, m, \mathsf{nonce}^*)$, and $\Pi_3(\mathtt{sid}, z^*)$. $\mathsf{FixTarget}$ enforces that this $\mathtt{sid}$ was selected for attack before any of those calls, and Comp_1 and $\mathsf{FixTarget}$ together enforce that $\mathtt{sid} \notin \mathsf{comp}_1$ at the time of Π_3. Therefore, the vk^*, m sent to Π_1 committed the adversary to the $F[\cdot, \cdot]$ position that would determine the z^* used to compute sas_1. Since Π_1 samples nonce uniformly at random, it follows by a one-time pad argument that sas_1 is uniformly random and unpredictable to the adversary before the time of Π_1 query. On the other hand, sas_2 is determined by $z = F[vk, m]$. The freshness of m and the assurance that $\mathsf{comp}_2 = \mathsf{F}$ at the time of Π_2 query together guarantee that $F[vk, m]$ (and therefore sas_2) is uniformly random and unpredictable to the adversary before the time of Π_2 query. Further, if the parties run on different messages m, then the random sas_1 and sas_2 thus determined are independent of one another, so the probability that they collide is $1/2^d$. Therefore, $\epsilon_2 \leq 1/2^d$.

Overall, we have $\epsilon_0 \leq 1/2^d + \varepsilon_{\mathsf{PVRF}}(\kappa)$. PVRF-SAS is secure.

1. for $i = 1, 2$ and all $\mathtt{sid}$, set $\mathsf{phase}_i^{\mathtt{sid}}$ to 0 and $\mathsf{m}_i^{\mathtt{sid}}, \mathsf{state}_i^{\mathtt{sid}}, \mathsf{sas}_i^{\mathtt{sid}}$ to $\perp$
2. set ATK, all $F[\cdot,\cdot]$ to $\perp$, $\mathsf{used}, \mathsf{comp}_1$ to $\emptyset$, and comp_2 to F
3. $vk \leftarrow \mathcal{A}(1^\kappa)$
4. $\mathtt{sid} \leftarrow \mathcal{A}^{\Pi_1,\Pi_2,\Pi_3,\mathsf{FixTarget},\mathsf{KeyComp},\mathsf{MalEval}}(vk)$
5. assert $\mathtt{sid} = \mathsf{ATK} \wedge \mathsf{phase}_1^{\mathtt{sid}} = 2 \wedge \mathsf{phase}_2^{\mathtt{sid}} = 1$
6. if $\mathsf{m}_1^{\mathtt{sid}} \neq \mathsf{m}_2^{\mathtt{sid}} \ \wedge \ \mathsf{sas}_1^{\mathtt{sid}} = \mathsf{sas}_2^{\mathtt{sid}}$ then return 1 else return 0

$\underline{\Pi_1(\mathtt{sid}, vk^*, m)}$:
assert $\mathsf{phase}_1^{\mathtt{sid}} = 0$
$(pm_1, \mathsf{st}_1^{\mathtt{sid}}) \leftarrow \mathsf{Init}(vk^*, m)$
$\mathsf{nonce} \leftarrow_\$ \{0,1\}^d$
$\mathsf{st}_1^{\mathtt{sid}} \leftarrow (vk^*, m, \mathsf{nonce})$
$\mathsf{phase}_1^{\mathtt{sid}} \leftarrow 1$, $\mathsf{m}_1^{\mathtt{sid}} \leftarrow m$
return (vt, nonce)

$\underline{\Pi_2(\mathtt{sid}, m, \mathsf{nonce}^*)}$:
assert $\mathsf{phase}_2^{\mathtt{sid}} = 0$ and $m \notin \mathsf{used}$
if $F[vk, m] = \perp$: $F[vk, m] \leftarrow_\$ \{0,1\}^d$
$z \leftarrow F[vk, m]$
$\mathsf{phase}_2^{\mathtt{sid}} \leftarrow 1$, $\mathsf{m}_2^{\mathtt{sid}} \leftarrow m$
$\mathsf{used} \leftarrow \mathsf{used} \cup \{m\}$
return $(z, z \oplus \mathsf{nonce}^*)$

$\underline{\mathsf{Comp}_2()}$:
assert $(\mathsf{ATK} \neq \perp \wedge \mathsf{sas}_1^{\mathsf{ATK}} \neq \perp \wedge \mathsf{sas}_2^{\mathsf{ATK}} \neq \perp)$
$\mathsf{comp}_2 \leftarrow \mathsf{T}$

$\underline{\Pi_3(\mathtt{sid}, z^*)}$:
assert $\mathsf{phase}_1^{\mathtt{sid}} = 1$
$(vk^*, m, \mathsf{nonce}) \leftarrow \mathsf{st}_1^{\mathtt{sid}}$
assert $F[vk^*, m] = z^*$ or $\mathtt{sid} \in \mathsf{comp}_1$
$\mathsf{phase}_2^{\mathtt{sid}} \leftarrow 2$
return $z^* \oplus \mathsf{nonce}$

$\underline{\mathsf{FixTarget}(\mathtt{sid})}$:
assert $\mathsf{ATK} = \perp$
assert $\mathsf{phase}_1^{\mathtt{sid}} = \mathsf{phase}_2^{\mathtt{sid}} = 0$
$\mathsf{ATK} \leftarrow \mathtt{sid}$

$\underline{\mathsf{Comp}_1(\mathtt{sid})}$:
assert $\mathsf{phase}_1^{\mathtt{sid}} \geq 1$
assert $(\mathsf{ATK} \neq \mathtt{sid} \vee (\mathsf{sas}_1^{\mathsf{ATK}} \neq \perp \wedge \mathsf{sas}_2^{\mathsf{ATK}} \neq \perp)$
$\mathsf{comp}_1 \leftarrow \mathsf{comp}_1 \cup \{\mathtt{sid}\}$

$\underline{\mathsf{MalEval}(vk^*, m)}$:
assert $(vk^* \neq vk \vee \mathsf{comp}_2 = \mathsf{T})$
if $F[vk^*, m] = \perp$: $F[vk^*, m] \leftarrow_\$ \{0,1\}^d$
return $F[vk^*, m]$

Fig. 12: Game G_2: Removing proofs from G_1

D Deniable Authenticated Key Exchange Against Malicious Accusers

In this section we provide the full specifications of strong deniability (i.e. deniability against malicious accusers), deferred from Section 4.2 to avoid redundancy. Figures 13 and 14 are the strong deniability game $\mathsf{Game}^{\mathsf{strong-ATK}}_{\mathcal{A},\mathsf{D}_1,\mathsf{D}_2,\mathcal{H},\mathcal{L}_{\mathsf{leak}},\mathcal{L}_{\mathsf{KDC}}}$ (parameterized by $\mathsf{leak}, \mathcal{L}_{\mathsf{KDC}}$, and ATK).

Definition 7. *(Deniability Against Malicious Accusers). Let $\mathcal{N} := [N]$ for $N \in \mathbb{N}$ be the set of all users, $\mathcal{H} \subseteq \mathcal{N}$ such that $\mathcal{H} \neq \varnothing$ be the set of accused users, and $\mathcal{C} := \mathcal{N} \setminus \mathcal{H}$ be the set of accusers.*

Protocol $\mathsf{BAKE}^\dagger$ *is* strongly deniable *with respect to leakage level* $\mathsf{leak} \in \{\mathsf{low}, \mathsf{med}, \mathsf{high}\}$*, KDC leakage bit* $\mathcal{L}_{\mathsf{KDC}} \in \{0,1\}$*, and locality* $\mathsf{ATK} \in \{\mathsf{local}, \mathsf{global}\}$ *if, for any efficient accuser* $(\mathcal{A}, \mathsf{D}_1)$*, there exists an efficient simulator* $\mathsf{Sim}_{\mathcal{A},\mathsf{D}_1} =$

$\mathsf{Game}^{\mathsf{strong-ATK}}_{\mathcal{A},\mathsf{D}_1,\mathsf{D}_2,\mathcal{H},\mathcal{L}_{\mathsf{leak}},\mathcal{L}_{\mathsf{KDC}}}(1^\kappa, \mathsf{mode})$:

$\mathcal{C} := \mathcal{N} \setminus \mathcal{H}$

for user $u \in \mathcal{N}$ **do**

  $(\mathsf{ipk}_u, \mathsf{isk}_u) \xleftarrow{\$} \mathsf{BAKE}^\dagger.\mathsf{IdKeyGen}(1^\kappa)$

  $\mathsf{counter}_u := 0$

  if $(u \in \mathcal{H})$ **then**

    $(\mathbf{prek}_u, \mathsf{st}_u) \xleftarrow{\$} \mathsf{BAKE}^\dagger.\mathsf{PreKeyBundleGen}(\mathsf{isk}_u)$

if $(\mathsf{mode} = \mathsf{real})$ **then**

  $((\mathbf{prek}_u)_{u\in\mathcal{C}}, \mathsf{st}_\mathcal{A}) \xleftarrow{\$} \mathcal{A}\left((\mathsf{ipk}_u)_{u\in\mathcal{N}}, (\mathsf{isk}_u)_{u\in\mathcal{C}}, (\mathbf{prek}_u)_{u\in\mathcal{H}}\right)$

else if $(\mathsf{mode} = \mathsf{sim})$ **then**

  $((\mathbf{prek}^*_u)_{u\in\mathcal{H}}, (\mathbf{prek}_u)_{u\in\mathcal{C}}, \mathsf{st}_{\mathsf{Sim}}) \xleftarrow{\$} \mathsf{SimPreK}_\mathcal{A}\left((\mathsf{ipk}_u)_{u\in\mathcal{N}}, (\mathsf{isk}_u)_{u\in\mathcal{C}}, (\mathbf{prek}_u)_{u\in\mathcal{H}}\right)$

  if $\neg\mathcal{L}_{\mathsf{KDC}}$ **then** $(\mathbf{prek}_u)_{u\in\mathcal{H}} \leftarrow (\mathbf{prek}^*_u)_{u\in\mathcal{H}}$

$\mathsf{st}_\mathsf{D} \xleftarrow{\$} \mathsf{D}_1^{\mathcal{O}_{\mathsf{ATK}}}((\mathsf{ipk}_u, \mathbf{prek}_u)_{u\in\mathcal{N}})$

if $(\mathsf{mode} = \mathsf{sim})$ **then**

  $\mathsf{st}_\mathcal{A} \xleftarrow{\$} \mathsf{SimSt}_\mathcal{A}(\mathsf{st}_{\mathsf{Sim}})$

$\mathsf{leak}_\mathsf{D} := \mathcal{L}_{\mathsf{leak}}((\mathsf{isk}_u, st_u)_{u\in\mathcal{H}})$

$b \xleftarrow{\$} \mathsf{D}_2(\mathsf{st}_\mathsf{D}, \mathsf{leak}_\mathsf{D}, \mathsf{st}_\mathcal{A})$

return b

Fig. 13: Local and global $\mathsf{BAKE}^\dagger$ deniability game for *malicious* accusers, with changes relative to BAKE deniability [20] marked in yellow.

Oracle $\mathcal{O}_{\mathsf{ATK}}(s, r)$:

assert $((s, r) \in \mathcal{N} \times \mathcal{N}) \wedge (s \neq r) \wedge ((s, r) \notin \mathcal{C} \times \mathcal{C})$
if $(\mathsf{ATK} = \mathsf{local})$ **then assert** $(s, r) \notin \mathcal{H} \times \mathcal{H}$
$\mathsf{counter}_r \leftarrow \mathsf{counter}_r + 1,\ t := \mathsf{counter}_r$
if $(t > L)$ **then** $t \leftarrow \perp$
if $(\mathsf{mode} = \mathsf{real})$ **then**
  if $(s \in \mathcal{H})$ **then**
    $(K, \rho, \mathsf{temp}) \xleftarrow{\$} \mathsf{BAKE}^{\dagger}.\mathsf{Send}(\mathsf{isk}_s, \mathsf{ipk}_r, \mathsf{prek}_{r,t})$
  else if $(s \in \mathcal{C})$ **then**
    $(\rho, \mathsf{st}_{\mathcal{A}}) \xleftarrow{\$} \mathcal{A}(\mathsf{st}_{\mathcal{A}}, (s, r, t))$
  if $(r \in \mathcal{H})$ **then**
    $(K', m, \mathsf{sas}_r, \mathsf{st}_r) \xleftarrow{\$} \mathsf{BAKE}^{\dagger}.\mathsf{Receive}(\mathsf{isk}_r, \mathsf{st}_r, \mathsf{ipk}_s, t, \rho)$
  else if $(r \in \mathcal{C})$ **then**
    $(m, \mathsf{sas}_r, \mathsf{st}_{\mathcal{A}}) \xleftarrow{\$} \mathcal{A}(\mathsf{st}_{\mathcal{A}}, \mathsf{ipk}_s, t, \rho)$
  if $(s \in \mathcal{H})$ **then**
    $\mathsf{sas}_s \leftarrow \mathsf{BAKE}^{\dagger}.\mathsf{SenderReceive}(\mathsf{temp}, m)$
  else if $(s \in \mathcal{C})$ **then**
    $(\mathsf{sas}_s, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(\mathsf{st}_{\mathcal{A}}, m)$
else if $(\mathsf{mode} = \mathsf{sim})$ **then**
  if $(s, r) \in \mathcal{H} \times \mathcal{C}$ **then**
    $(K, \rho, m, \mathsf{sas}_s, \mathsf{st}_{\mathsf{Sim}}) \xleftarrow{\$} \mathsf{SimTrans}_{\mathcal{A}}(\mathsf{isk}_r, \mathsf{st}_{\mathsf{Sim}}, (s, r, t))$
  else if $(s, r) \in \mathcal{C} \times \mathcal{H}$ **then**
    $(K', \rho, m, \mathsf{sas}_r, \mathsf{st}_{\mathsf{Sim}}) \xleftarrow{\$} \mathsf{SimTrans}_{\mathcal{A}}(\mathsf{isk}_s, \mathsf{st}_{\mathsf{Sim}}, (s, r, t))$
    $(K'', \cdot, \cdot, \cdot) \xleftarrow{\$} \mathsf{BAKE}^{\dagger}.\mathsf{Receive}(\mathsf{isk}_r, \mathsf{st}_r, \mathsf{ipk}_s, t, \rho)$
    if $(K' \neq \perp)$ **then** $K' \leftarrow K''$
    $\mathsf{st}_r \xleftarrow{\$} \mathsf{SimSt}_{\mathcal{H}}(\mathsf{isk}_r, \mathsf{st}_r)$
  else if $(s, r) \in \mathcal{H} \times \mathcal{H}$ **then**
    $(K, K', \rho, m, \mathsf{sas}_s, \mathsf{sas}_r, \mathsf{st}_{\mathsf{Sim}}) \xleftarrow{\$} \mathsf{SimTrans}_{\mathcal{A}}(\perp, \mathsf{st}_{\mathsf{Sim}}, (s, r, t))$
    $\mathsf{st}_r \xleftarrow{\$} \mathsf{SimSt}_{\mathcal{H}}(\mathsf{isk}_r, \mathsf{st}_r)$
if $(s, r) \in \mathcal{H} \times \mathcal{C}$ **then return** $(K, \rho, m, \mathsf{sas}_s)$
else if $(s, r) \in \mathcal{C} \times \mathcal{H}$ **then return** $(K', \rho, m, \mathsf{sas}_r)$
else return $(K, K', \rho, m, \mathsf{sas}_s, \mathsf{sas}_r)$

Fig. 14: Oracle $\mathcal{O}_{\mathsf{ATK}}$ for malicious accusers in $\mathsf{BAKE}^{\dagger}$ deniability game in Fig. 13, with changes from BAKE deniability game [20] marked in yellow.

$(\mathsf{SimPreK}_{\mathcal{A},\mathsf{D}_1}, \mathsf{SimTrans}_{\mathcal{A},\mathsf{D}_1}, \mathsf{SimSt}_{\mathcal{H}}, \mathsf{SimSt}_{\mathcal{A},\mathsf{D}_1})$ *such that for any efficient distinguisher* D_2 *we have*

$$\Pr\left[\mathsf{Game}^{\mathsf{strong-local}}_{\mathcal{A},\mathsf{D}_1,\mathsf{D}_2,\mathcal{H},\mathcal{L}_{\mathsf{leak}},\mathcal{L}_{\mathsf{KDC}}}(1^\kappa,\mathsf{real})=0\right] \leq \Pr\left[\mathsf{Game}^{\mathsf{strong-local}}_{\mathcal{A},\mathsf{D}_1,\mathsf{D}_2,\mathcal{H},\mathcal{L}_{\mathsf{leak}},\mathcal{L}_{\mathsf{KDC}}}(1^\kappa,\mathsf{sim})=0\right]+\epsilon(\kappa)$$

for some negligible function ϵ*, where* $\mathsf{Game}^{\mathsf{strong-local}}_{\mathcal{A},\mathsf{D}_1,\mathsf{D}_2,\mathcal{H},\mathcal{L}_{\mathsf{leak}},\mathcal{L}_{\mathsf{KDC}}}$ *is given in Figures 13 and 14.*

Online Deniability and an Attack on the Model of [20]. As explained in Section 1, X3DH is not deniable in a model where the distinguisher D is allowed "online" communication with the malicious accuser $\mathcal{A}$ without any restrictions.

In the strong deniability model of [20], online communication from $\mathcal{A}$ to D is not difficult: D can simply run $\mathcal{A}$ with any counterparty, and $\mathcal{A}$ can output an arbitrary message into the resulting transcript via ρ or m. Seemingly, however, online communication is still prevented because there is no way for D to send a message to $\mathcal{A}$. Surprisingly, though, there is actually a subtle "back-channel" by which D can pass information to $\mathcal{A}$. The distinguisher can use their choice of (s, r) parties sent to oracle $\mathcal{O}_{\mathsf{ATK}}$ as a way of encoding an arbitrary bitstring.

For example, consider an accuser $\mathcal{A}$ who, on the first key exchange session, outputs the public keys $(\mathsf{ipk}_h, \mathbf{prek_h})$ of some honest party h into the session transcript. The distinguisher, after seeing these keys, forms a sender protocol message ρ and uses the next thousand (s, r) pairs to encode it. $\mathcal{A}$ reconstructs this ρ, and, on the next session, passes it to honest receiver h. This honest receiver unwittingly completes a key exchange with D, and no simulator can simulate the resulting K'.

This back-channel results in an explicit attack on the strong deniability of X3DH, in contradiction to the positive results of [20]. Katsumata et al. prove strong deniability in the GGM, assuming that $\mathcal{A}$ is a generic algorithm and therefore the simulator, by observing all generic group operations of $\mathcal{A}$, can extract the discrete logarithm of any group element output by $\mathcal{A}$ (or observe that the group element is derived from a random oracle hash and therefore its discrete logarithm is unknown to all parties). This argument is predicated on the implicit assumption that $\mathcal{A}$ has no source for group elements other than those it computes itself or derives from a random oracle hash, and it therefore fails to account for the possibility of a group element passed from D to $\mathcal{A}$ via the hidden back-channel.

We stress that this attack is highly artificial and does not reveal anything meaningful about the real deniability of X3DH. Rather, it is an attack on the model of [20] that serves to underscore the impossibility of online deniability for X3DH. Even the subtlest channel of D-to-$\mathcal{A}$ communication breaks strong deniability.

To repair the model of [20], we propose to split D into two algorithms, D_1 that directs parties to perform key exchanges via the $\mathcal{O}_{\mathsf{ATK}}$ oracle, and D_2 that ultimately attempts to distinguish the real and simulated worlds. We modify the definition of strong deniability so that it is for every $(\mathcal{A}, \mathsf{D}_1)$ pair that there

must exist a simulator. Essentially, we treat D_1 as an appendage of the accuser, whereas [20] treats it as an appendage of the judge.

E Proofs of X3DH+PVRF-SAS Security

In this section we prove Theorems 3 and 4, i.e. that protocol X3DH+PVRF-SAS is secure in spite of key reuse between X3DH and PECVRF.

We begin by proving Theorem 3, which states that X3DH with PECVRF key reuse is a key-indistinguishable bundled authenticated key exchange.

Proof. To demonstrate the security of X3DH, [18] define the notion of BAKE key indistinguishability and prove that X3DH satisfies it (under GapDH and in the ROM). The definition of this property is long and complex, involving many possible cases corresponding to different combinations of keys that may be compromised. We refer to [18, Definition 3] for the full details, and here we only use the key indistinguishability game in a "black-box" way. Essentially, we only argue that the reuse of X3DH keys for PECVRF evaluation does not invalidate the existing security argument.

(1) Denote the X3DH key indistinguishability game as G. Consider a variation G^* of G where, after receiving any public key, the adversary $\mathcal{A}$ has oracle access to arbitrary PECVRF evaluations under it. Clearly, if X3DH is secure in this game, then it is secure when its keys are reused as in X3DH+PVRF-SAS. Suppose there exists an adversary $\mathcal{A}$ with non-negligible advantage ϵ in G^*. We will construct adversary $\mathcal{A}'$ with non-negligible advantage ϵ' in G.

(2) We begin by decoupling G^* into G and PECVRF. Whenever G needs to sample a new key, we perform PECVRF.KeyGen followed immediately by PECVRF.KeyComp and send the public key and compromised secret key to G. $\mathcal{A}$ interacts with the interface of G^* and its PECVRF evaluation queries are routed to the relevant PECVRF instance. These changes are purely conceptual and do not change the view of $\mathcal{A}$ at all.

(3) We next observe that G and $\mathcal{A}$ together form an environment with respect to PECVRF. In particular, we define the output of this environment as the output of the key indistinguishability game G. By the UC security of PECVRF (Theorem 7) there exists an efficient simulator $\mathsf{SIM}_{\mathsf{PECVRF}}$ such that no efficient environment can distinguish $\mathsf{SIM}_{\mathsf{PECVRF}}$ in interaction with $\mathcal{F}_{\mathsf{PVRF}}$ from the real PECVRF protocol with probability greater than negligible $\epsilon_{\mathsf{PECVRF}}$ (under the DDH assumption and in the ROM). We therefore replace PECVRF with $\mathsf{SIM}_{\mathsf{PECVRF}} \leftrightarrow \mathcal{F}_{\mathsf{PVRF}}$ to create a new game in which the output of G is negligibly different from before.

(4) In order to construct an adversary $\mathcal{A}'$ against G we must allow G to sample its keys internally, not take them as input from $\mathsf{SIM}_{\mathsf{PECVRF}}$. By inspection, we observe that $\mathsf{SIM}_{\mathsf{PECVRF}}$ (Figure 8) never makes use of secret keys (until the time that a key is adaptively corrupted). We therefore need only slightly modify $\mathsf{SIM}_{\mathsf{PECVRF}}$ so that it runs on public keys received from G rather than sampling them itself. Denote this modified simulator $\mathsf{SIM}'_{\mathsf{PECVRF}}$, and define $\mathcal{A}'$ to be the combination of $\mathcal{A}$, $\mathsf{SIM}'_{\mathsf{PECVRF}}$, and $\mathcal{F}_{\mathsf{PVRF}}$ in interaction with each other.

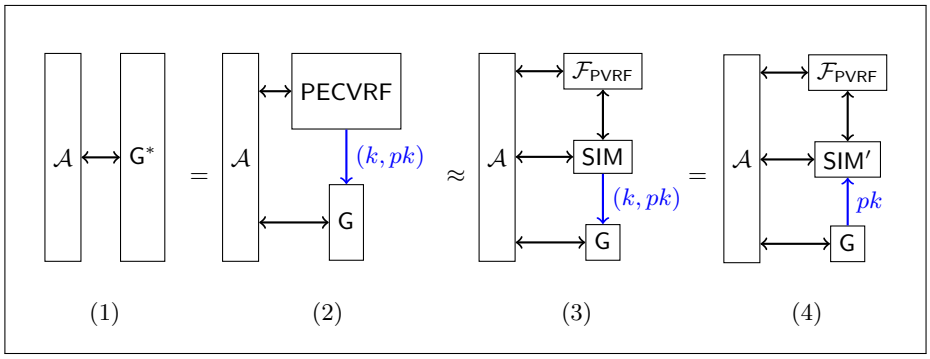

Fig. 15: Diagram of proof that X3DH with PECVRF key reuse is key-indistinguishable

The argument above is diagrammed in Figure 15. All together, we have $\epsilon \leq \epsilon' + \epsilon_{\mathsf{PECVRF}}$. Under GapDH and in the ROM, ϵ' is negligible per [18, Theorem 6], and, under DDH and in the ROM, $\epsilon_{\mathsf{PECVRF}}$ is negligible per Theorem 7. Therefore, ϵ is negligible, and X3DH+PVRF-SAS is key-indistinguishable.

Theorem 3 is proven.

We next prove Theorem 4, which states that PECVRF with X3DH key reuse realizes $\mathcal{F}_{\mathsf{PVRF}}$.

Proof. We define PECVRF* to be an augmented version of protocol PECVRF wherein the ideal adversary $\mathcal{A}^*$ is granted access to an "X3DH oracle". In particular, $\mathcal{A}^*$ can use this oracle to either (1) perform an X3DH authenticated key exchange between 2 honest parties using any set of PECVRF keys or (2) perform an X3DH authenticated key exchange between an honest party using any set of PECVRF keys and $\mathcal{A}^*$ using arbitrary (adversarial) public keys. Notice that, in X3DH, protocol messages only consist of public keys and the key confirmation τ_{conf}. In PECVRF*, the public keys are already known to $\mathcal{A}^*$, and so we omit them from the output of our new X3DH oracle. Instead, the X3DH oracle returns to $\mathcal{A}^*$ $(K||\tau_{\mathsf{conf}})$ for each honest party, i.e. the output of the key derivation function KDF. Sending K to $\mathcal{A}^*$ models the fact that session keys are subject to arbitrary leakage after key exchange completion.

In X3DH+PVRF-SAS, only one-time keys otsk (and semi-static keys ssk if one-time keys run out) are reused as PECVRF keys. Thus, PECVRF* is a generalization of X3DH+PVRF-SAS wherein all X3DH keys are potentially reused for PECVRF. If PECVRF* UC-realizes $\mathcal{F}_{\mathsf{PVRF}}$, then PECVRF is secure when its keys are reused as in X3DH+PVRF-SAS.

To prove PECVRF* security we propose a minor modification to the proof of PECVRF security (Theorem 7). In particular, note that simulation of the X3DH oracle of PECVRF* is easily added to $\mathsf{SIM}_{\mathsf{PECVRF}}$ (Figure 8) because $\mathsf{SIM}_{\mathsf{PECVRF}}$ samples and knows all honest parties' PECVRF secret keys k. Thus, the simulator can simply run the real X3DH code on behalf of honest parties and return the output of KDF. Denote by $\mathsf{SIM}^*_{\mathsf{PECVRF}}$ this augmented version of $\mathsf{SIM}_{\mathsf{PECVRF}}$.

By inspection, almost the entirety of the proof of Theorem 7 applies without modification to proving that $\mathsf{SIM}^*_{\mathsf{PECVRF}}$ is a simulator for PECVRF^*. The code of the X3DH oracle is identical in the real world of PECVRF^* and in the simulated world of $\mathsf{SIM}^*_{\mathsf{PECVRF}}$. There is, however, one step in the proof where secret keys k are not available—namely, the CDH reduction. That reduction derives all public keys from CDH challenges rather than sampling them internally. Thus, there is no obvious way for this reduction to emulate the X3DH oracle.

To resolve this issue, we must "upgrade" to a GapDH reduction wherein the reduction has access to a DDH oracle. Since the reduction does not hold honest parties' secret keys, it cannot compute $(K||\tau_{\mathsf{conf}})$ by running the real X3DH code. Instead, the reduction simply samples random $(K||\tau_{\mathsf{conf}})$ and returns it to $\mathcal{A}^*$. If KDF is modeled as a random oracle (as it is in the proof of X3DH security [18]), then the environment cannot distinguish this behavior from real unless it queries KDF on the input that the real honest party would have queried it on. The reduction monitors the environment's KDF queries and programs output $(K||\tau_{\mathsf{conf}})$ matching the output of the X3DH oracle if KDF is queried on an input that matches an X3DH session. To compute the KDF input for an X3DH session requires computing the Diffie-Hellman function on 4 pairs of public keys. The reduction cannot make this computation, but it can use the DDH oracle to check whether a provided input is a match.

In spite of key reuse with X3DH, protocol PECVRF^* realizes $\mathcal{F}_{\mathsf{PVRF}}$.

F Deniability of X3DH and X3DH+VRF

In the following we will use the model from [18]. Extending the model to also have a message flow from the receiver to the sender. The honest but curious accuser deletes their ephemeral values.

F.1 Deniability of X3DH against passive adversaries

First we discuss deniability of X3DH against honest-but-curious accusers. Protocol X3DH is fully specified as a BAKE scheme in Figure 16.

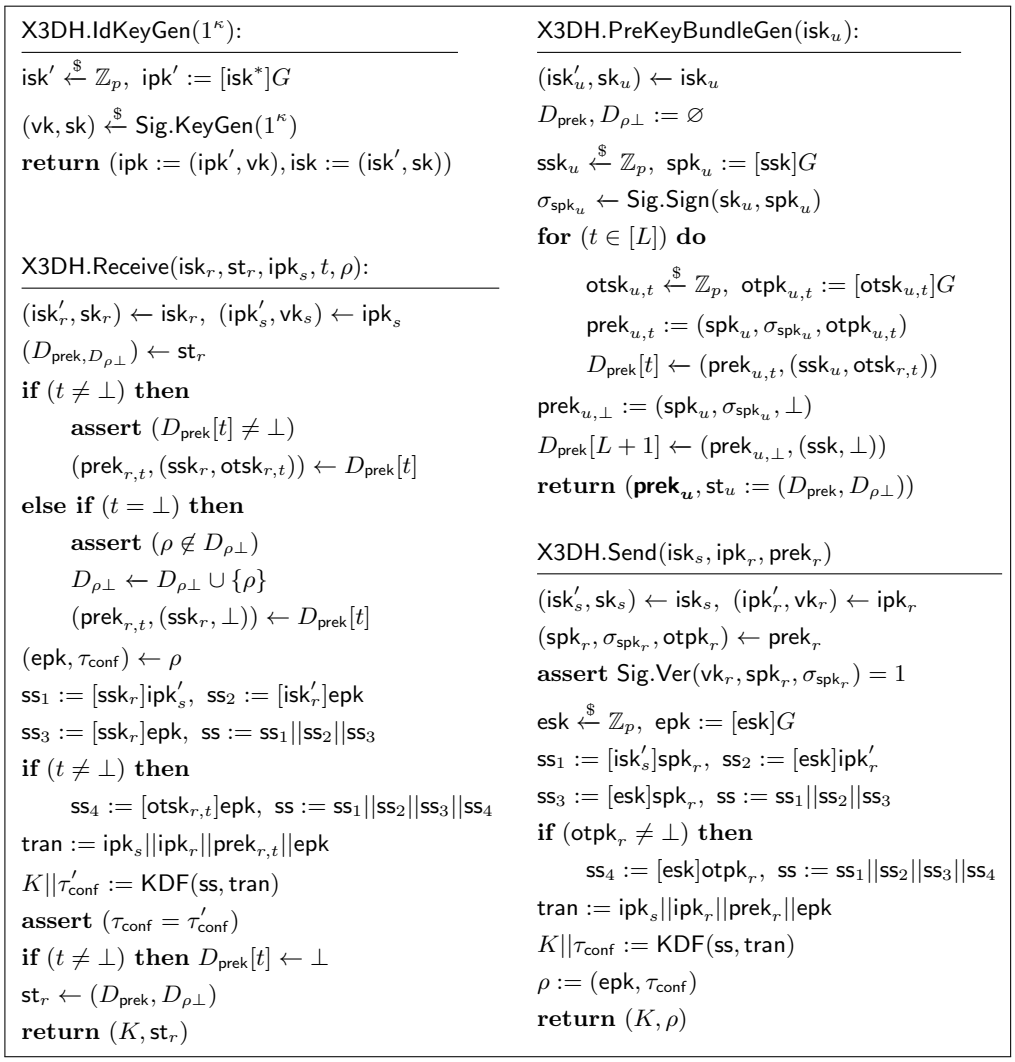

Fig. 16: BAKE protocol X3DH = (IdKeyGen, PreKeyBundleGen, Send, Receive)

Lemma 1 ([20]). *The* X3DH *protocol is* locally deniable *with respect to leakage function* $\mathcal{L}_{\mathsf{high}}$ *disclosure function* $\mathcal{D}_{\mathsf{high}}$ *and KDC leakage bit* $\mathcal{L}_{\mathsf{KDC}} = 1$.

We will define the simulators here. For the proof refer to [20].

$\mathsf{SimPreK}\left((\mathsf{ipk}_u)_{u\in\mathcal{N}}, (\mathsf{isk}_u)_{u\in\mathcal{C}}, (\mathbf{prek_u})_{u\in\mathcal{H}}\right)$: For $u \in \mathcal{C}$ run $(\mathbf{prek_u}, \mathsf{st}_u) \leftarrow \mathsf{X3DH.PreKeyBundleGen}(\mathsf{isk}_u)$, and output $((\mathbf{prek_u})_{u\in\mathcal{H}}, (\mathbf{prek_u}, \mathsf{st}_u)_{u\in\mathcal{C}}, \mathsf{st}_{\mathsf{Sim}})$ with $\mathsf{st}_{\mathsf{Sim}} = ((\mathsf{ipk}_u, \mathbf{prek_u})_{u\in\mathcal{N}}, (\mathsf{isk}_u, \mathsf{st}_u)_{u\in\mathcal{C}})$. Note the simulator does not overwrite any prekey bundles.
$\mathsf{SimTrans}(\mathsf{isk}, \mathsf{st}_{\mathsf{Sim}}, (s, r, t))$: Here isk is either isk_s,isk_r or $\perp$ depending on $s \in \mathcal{C},\ r \in \mathcal{C}$ or $(s, r) \in \mathcal{H} \times \mathcal{H}$, respectively.

If $(s, r) \in \mathcal{H} \times \mathcal{C}$: First sample $\mathsf{esk} \xleftarrow{\$} \mathbb{Z}_p$ and set $\mathsf{epk} := [\mathsf{esk}]G$. Then extract the signed prekey ssk_r from $\mathsf{st}_r \in \mathsf{st}_{\mathsf{Sim}}$ and compute

$$(\mathsf{ss}_1, \mathsf{ss}_2, \mathsf{ss}_3, \mathsf{ss}_4) := \left([\mathsf{ssk}_r]\mathsf{ipk}'_s, [\mathsf{esk}]\mathsf{ipk}'_r, [\mathsf{esk}]\mathsf{spk}_r, [\mathsf{esk}]\mathsf{otpk}_{r,t}\right)$$

compute the rest of X3DH.Send as in Fig.16 and output (K, ρ).
If $(s, r) \in \mathcal{C} \times \mathcal{H}$: Simply run $(K, \rho) \leftarrow \mathsf{X3DH.Send}(\mathsf{isk}_s, \mathsf{ipk}_r, \mathsf{prek}_{r,t})$ and output (K, ρ).

$\mathsf{SimSt}_{\mathcal{H}}(\mathsf{isk}_r, \mathsf{st}_r)$: Parse $(D_{\mathsf{prek}}, D_{\rho\perp}) \leftarrow \mathsf{st}_r$ and delete the last unused one-time state, i.e., let t be the smallest index s.t. $D_{\mathsf{prek}}[t] \neq \perp$. If $t \leq L$, let $D_{\mathsf{prek}}[t] \leftarrow \perp$. Output the updated state st_r.
$\mathsf{SimSt}_{\mathcal{C}}(\mathsf{st}_{\mathsf{Sim}})$: For $u \in \mathcal{C}$ extract st_u from $\mathsf{st}_{\mathsf{Sim}}$, parse $(D_{\mathsf{prek}}, D_{\rho\perp}) \leftarrow \mathsf{st}_u$ and delete the ephemeral secrets associated to used prekeys, i.e., for $t \in [\mathsf{counter}_u]$, with $t \leq L$, let $D_{\mathsf{prek}}[t] \leftarrow \perp$. Output $(st_u)_{u\in\mathcal{C}}$.

Lemma 2 ([20]). *Assume* KDF *is modeled as a random oracle and the* DDH *assumption holds. Then* X3DH *is globally deniable for honest users* $\mathcal{H} \subseteq \mathcal{N}$ *against honest-but-curious accusers* $\mathcal{C} := \mathcal{N} \backslash \mathcal{H}$ *with respect to disclosure function* $\mathcal{D}_{\mathsf{high}}$*, KDC leakage bit* $\mathcal{L}_{\mathsf{KDC}} = 1$ *and leakage function* $\mathcal{L}_{\mathsf{leak}}$ *for* $\mathsf{leak} = \mathsf{high}$ *if the one-time prekey bundles are never depleted; and for* $\mathsf{leak} = \mathsf{med}$ *otherwise.*

The simulator here is defined identically to Lemma 1 except we have to further specify the description of SimTrans on input $(s, r) \in \mathcal{H} \times \mathcal{H}$.

$\mathsf{SimTrans}(\mathsf{st}_{\mathsf{Sim}}, (s, r, t))$: Sample $\mathsf{epk} \xleftarrow{\$} \mathbb{G}$, sample random $K||\tau_{\mathsf{conf}}$ from range of KDF. Compute ρ as in Fig.16 and output (K, K, ρ).

F.2 Strong Deniability of X3DH

Lemma 3 ([20]). *Assume the underlying group used by* X3DH *is a generic group of prime order p, and the key derivation function* KDF *is modeled as a random oracle. Then* X3DH *is strongly local deniable for honest users* $\mathcal{H} \subseteq \mathcal{N}$ *against malicious accusers* $\mathcal{C} := \mathcal{N} \setminus \mathcal{H}$ *with respect to leakage function* $\mathcal{L}_{\mathsf{high}}$ *and KDC leakage bit* $\mathcal{L}_{\mathsf{KDC}} = 1$.

Lemma 4 ([20]). *Assume the underlying group used by* X3DH *is a generic group of prime order p and* KDF *is modeled as a random oracle. Then* X3DH *is strongly global deniable for honest users $\mathcal{H}$ against malicious accusers $\mathcal{C}$ with respect to KDC leakage it $\mathcal{L}_{\mathsf{KDC}} = 1$ and leakage function $\mathcal{L}_{\mathsf{leak}}$ for $\mathsf{leak} = \mathsf{high}$ if the one-time prekeys are never depleted and for $\mathsf{leak} = \mathsf{med}$ otherwise.*

In GGM when $\mathcal{A}$ outputs a label s, $\mathsf{Sim}_{\mathcal{A}}$ can efficiently verify if s is a linear combination of labels/group elements (or obliviously sampled labels/group elements) i.e. whether $\mathcal{A}$ knows the discrete log of s to the given random injection function/generator. If it does not, s is a Bad_L label. Otherwise it is a Good_L label.

$\mathsf{SimPreK}_{\mathcal{A}}\left((\mathsf{ipk}_u)_{u\in\mathcal{N}}, (\mathsf{isk}_u)_{u\in\mathcal{C}}, (\mathbf{prek}_{\boldsymbol{u}})_{u\in\mathcal{H}}\right)$: It first internally executes the adversary $((\mathbf{prek}_{\boldsymbol{u}})_{u\in\mathcal{C}}, \mathsf{st}_{\mathcal{A}}) \xleftarrow{\$} \mathcal{A}((\mathsf{ipk}_u)_{u\in\mathcal{N}}, (\mathsf{isk}_u)_{u\in\mathcal{C}}, (\mathbf{prek}_{\boldsymbol{u}})_{u\in\mathcal{H}})$. It then outputs
$((\mathbf{prek}_{\boldsymbol{u}})_{u\in\mathcal{H}}, (\mathbf{prek}_{\boldsymbol{u}})_{u\in\mathcal{C}}, \mathsf{st}_{\mathcal{A}}, \mathsf{st}_{\mathsf{Sim}})$ with $\mathsf{st}_{\mathsf{Sim}} = ((\mathsf{ipk}_u, \mathbf{prek}_{\boldsymbol{u}})_{u\in\mathcal{N}}, \mathsf{st}_{\mathcal{A}}, \mathsf{aux}_{\mathsf{Sim}})$
where $\mathsf{aux}_{\mathsf{Sim}}$ contains all GGM and random oracle queries made by $\mathcal{A}$ so far.
$\mathsf{SimTrans}_{\mathcal{A}}(\mathsf{isk}, \mathsf{st}_{\mathsf{Sim}}, (s, r, t))$: Here isk is either isk_s,isk_r or $\perp$ depending on $s \in \mathcal{C},\ r \in \mathcal{C}$ or $(s, r) \in \mathcal{H} \times \mathcal{H}$, respectively.

If $(s, r) \in \mathcal{H} \times \mathcal{C}$: First retrieve the prekey bundle $(\mathsf{spk}_r, \sigma_{\mathsf{spk}_r}, \mathsf{otpk}_{r,t}) \leftarrow \mathsf{prek}_{r,t}$ of the malicious receiver, sample $\mathsf{esk} \xleftarrow{\$} \mathbb{Z}_p$ and query the GGM labeling oracle to receive $\Xi(\mathsf{esk})$. Then check if spk_r is a Good_L using $\mathsf{aux}_{\mathsf{Sim}} \in \mathsf{st}_{\mathsf{Sim}}$. If so, compute the unique signed prekey secret $\mathsf{ssk}_r \in \mathbb{Z}_p$ such that $\Xi(\mathsf{ssk}_r) = \mathsf{spk}_r$. Query the GGM labeling oracle on tuples $(\mathsf{ipk}'_s, \mathsf{ssk}_r), (\mathsf{ipk}'_r, \mathsf{esk}), (\mathsf{spk}_r, \mathsf{esk})$ and $(\mathsf{otpk}_{r,t}, \mathsf{esk})$ to obtain labels (s_1, s_2, s_3, s_4). Set

$$(\mathsf{ss}_1, \mathsf{ss}_2, \mathsf{ss}_3, \mathsf{ss}_4) = (s_1, s_2, s_3, s_4)$$

compute K and ρ according to Fig.16 and output $(K, \rho, \mathsf{st}_{\mathsf{Sim}})$. Note above if $\mathsf{otpk}_{r,t} = \perp$ then Sim simply ignores generation of s_4.
Otherwise, if spk_r is a Bad_L label then sample random $K||\tau_{\mathsf{conf}}$ from range of KDF, compute ρ and output $(K, \rho, \mathsf{st}_{\mathsf{Sim}})$.

If $(s, r) \in \mathcal{C} \times \mathcal{H}$: First execute the malicious sender internally $(\rho, \mathsf{st}_{\mathcal{A}}) \xleftarrow{\$} \mathcal{A}(\mathsf{isk}_s, \mathsf{st}_{\mathcal{A}}, (s, r, t))$ using the accuser's state $\mathsf{st}_{\mathcal{A}} \in \mathsf{st}_{\mathsf{Sim}}$. The simulator updates $\mathsf{st}_{\mathsf{Sim}}$ by storing the updated $\mathsf{st}_{\mathcal{A}}$. Parse $(\mathsf{epk}, \tau_{\mathsf{conf}}) \leftarrow \rho$ and check if $\mathcal{A}$ ever made a random oracle query on $(\mathsf{ss}, \mathsf{tran})$ where $\mathsf{tran} := \mathsf{ipk}_s||\mathsf{ipk}_r||\mathsf{prek}_{r,t}||\mathsf{epk}$ such that $K||\tau_{\mathsf{conf}} = \mathsf{KDF}(\mathsf{ss}, \mathsf{tran})$ using $\mathsf{aux}_{\mathsf{Sim}}$. If not, output $(\perp, \rho, \mathsf{st}_{\mathsf{Sim}}$. Otherwise, retrieve the first such tuple (ss, K) and output $(K, \rho, \mathsf{st}_{\mathsf{Sim}})$.

If $(s, r) \in \mathcal{H} \times \mathcal{H}$: First sample $\mathsf{esk} \xleftarrow{\$} \mathbb{Z}_p$ and query the GGM labeling oracle to receive $\mathsf{epk} := \Xi(\mathsf{esk})$. Then sample random $K||\tau_{\mathsf{conf}}$ from the range of KDF and compute ρ. Output (K, K, ρ).

$\mathsf{SimSt}_{\mathcal{H}}$ Parse $(D_{\mathsf{prek}}, D_{\rho\perp}) \leftarrow \mathsf{st}_r$ and delete the last unused one-time state, i.e., let t be the smallest index s.t. $D_{\mathsf{prek}}[t] \neq \perp$. If $t \leq L$, let $D_{\mathsf{prek}}[t] \leftarrow \perp$. Output the updated state st_r.
$\mathsf{SimSt}_{\mathcal{A}}(\mathsf{st}_{\mathsf{Sim}})$: Extract $\mathsf{st}_{\mathcal{A}}$ from $\mathsf{st}_{\mathsf{Sim}}$ and output $\mathsf{st}_{\mathcal{A}}$.

F.3 Deniability of X3DH + VRF against passive adversaries

Consider the protocol from [17] shown in Fig.17. The parts that differ from regular X3DH are highlighted in yellow.

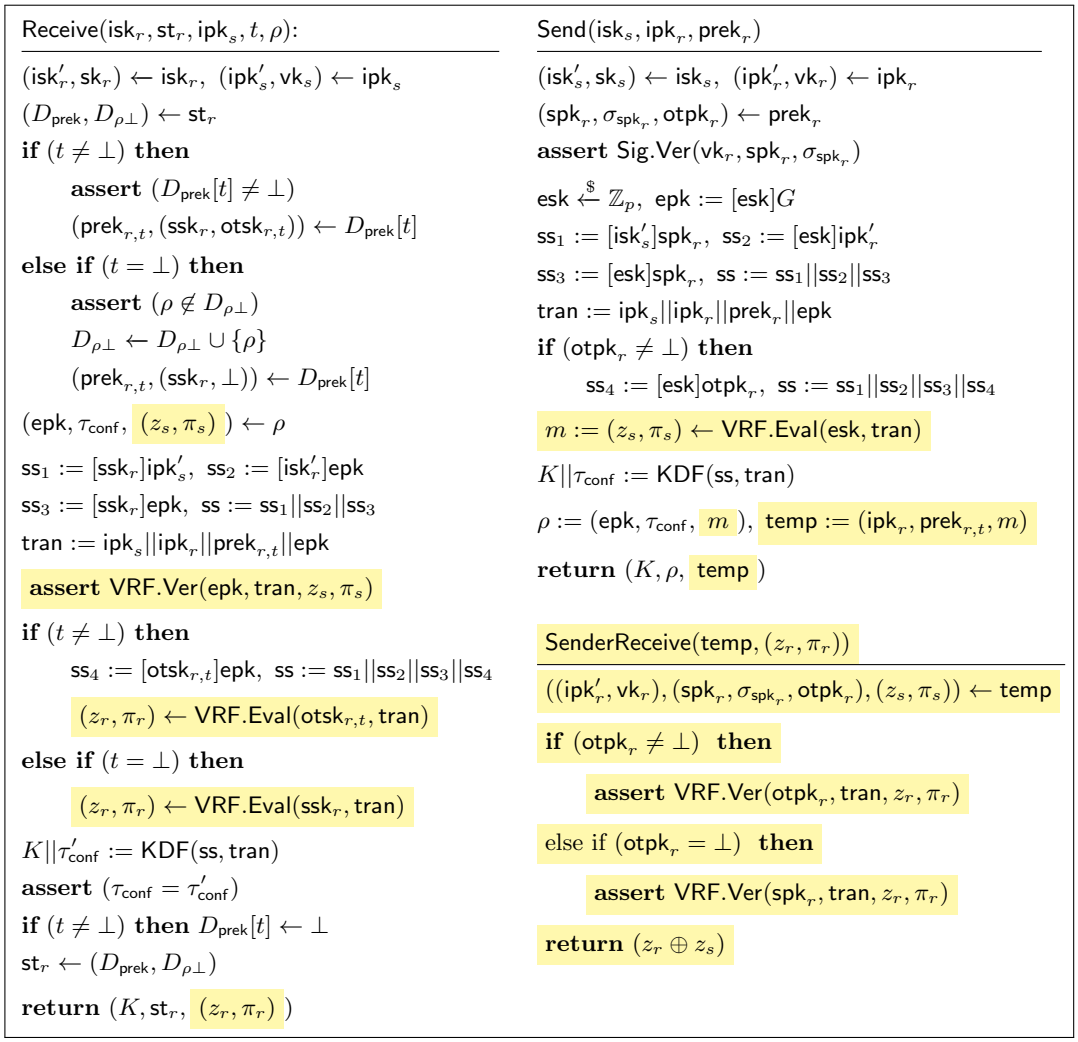

Fig. 17:
X3DH + VRF = (IdKeyGen, PreKeyBundleGen, Send, Receive, SenderReceive) which uses VRF = (KeyGen, Eval, Ver). Differences from X3DH in Fig.16 are highlighted in yellow.

Lemma 5. *The* X3DH + VRF *protocol is globally deniable with respect to disclosure function* $\mathcal{D}_{\mathsf{high}}$*, KDC leakage bit* $\mathcal{L}_{\mathsf{KDC}} = 0$ *leakage function* $\mathcal{L}_{\mathsf{high}}$ *if the one-time prekey bundles are never depleted; and it is not locally deniable for the receiver or globally deniable for the sender and receiver otherwise. It is still deniable for honest senders for leakage function* $\mathcal{L}_{\mathsf{high}}$ *even if the one-time prekey bundles are depleted.*

Following is the simulator.

$\mathsf{SimPreK}\,((\mathsf{ipk}_u)_{u\in\mathcal{N}}, (\mathsf{isk}_u)_{u\in\mathcal{C}}, (\mathbf{prek_u})_{u\in\mathcal{H}})$: For $u \in \mathcal{C}$ run $(\mathbf{prek_u}, \mathsf{st}_u) \leftarrow \mathsf{PreKeyBundleGen}(\mathsf{isk}_u)$, but additionally parse each prekey for each honest user $(\mathsf{spk}_u, \sigma_{\mathsf{spk}_u}, \mathsf{otpk}_{u,t}) \leftarrow \mathsf{prek}_{u,t}$. Since $\mathcal{L}_{\mathsf{KDC}} = 0$ we can overwrite parts of the prekey bundle for honest users. Sample new random one-time secret keys $\mathsf{otsk}^*_{r,t} \xleftarrow{\$} \mathbb{Z}_p$ and compute $\mathsf{otpk}^*_{u,t} := [\mathsf{otsk}^*_{u,t}]G$. Replace the honest one-time prekeys with simulated one-time prekeys $\mathsf{prek}^*_{u,t} \leftarrow (\mathsf{spk}_u, \sigma_{\mathsf{spk}_u}, \mathsf{otpk}^*_{r,t})$. Output $((\mathbf{prek^*_u})_{u\in\mathcal{H}}, (\mathbf{prek_u}, \mathsf{st}_u)_{u\in\mathcal{C}}, \mathsf{st}_{\mathsf{Sim}})$ with $\mathsf{st}_{\mathsf{Sim}} = ((\mathsf{ipk}_u, \mathbf{prek_u})_{u\in\mathcal{N}}, (\mathsf{isk}_u, \mathsf{st}_u)_{u\in\mathcal{C}}, (\mathbf{otsk^*_u})_{u\in\mathcal{H}})$.

$\mathsf{SimTrans}(\mathsf{isk}, \mathsf{st}_{\mathsf{Sim}}, (s, r, t))$: Here isk is either isk_s,isk_r or $\perp$ depending on $s \in \mathcal{C},\ r \in \mathcal{C}$ or $(s, r) \in \mathcal{H} \times \mathcal{H}$, respectively.

If $(s, r) \in \mathcal{H} \times \mathcal{C}$: First sample $\mathsf{esk} \xleftarrow{\$} \mathbb{Z}_p$ and set $\mathsf{epk} := [\mathsf{esk}]G$. Then extract the signed prekey ssk_r from $\mathsf{st}_r \in \mathsf{st}_{\mathsf{Sim}}$ and compute

$$(\mathsf{ss}_1, \mathsf{ss}_2, \mathsf{ss}_3, \mathsf{ss}_4) := \left([\mathsf{ssk}_r]\mathsf{ipk}'_s, [\mathsf{esk}]\mathsf{ipk}'_r, [\mathsf{esk}]\mathsf{spk}_r, [\mathsf{esk}]\mathsf{otpk}_{r,t}\right)$$

compute the rest of Send as in Fig.17 to get (K, ρ, temp). Extract $\mathsf{st}_r \in \mathsf{st}_{\mathsf{Sim}}$ from the *honest-but-curious* accuser and compute $(K', \mathsf{st}_r, m) \xleftarrow{\$} \mathsf{Receive}(\mathsf{isk}_r, \mathsf{st}'_r, \mathsf{ipk}_s, t, \rho)$ where st'_r is a copy of st_r since $\mathsf{SimSt}_{\mathcal{C}}$ updates the states of accusing receivers. Run $\mathsf{sas}_r \leftarrow \mathsf{SenderReceive}(\mathsf{temp}, m)$. Output (K, ρ, m).

If $(s, r) \in \mathcal{C} \times \mathcal{H}$: Simply run $(K, \rho, \mathsf{temp}) \leftarrow \mathsf{Send}(\mathsf{isk}_s, \mathsf{ipk}_r, \mathsf{prek}_{r,t})$.

If we assume one-time prekeys never run out then the VRF is always computed under the one-time prekey $\mathsf{otpk}_{r,t}$. In the simulated world these values can be replaced indistinguishably by the simulator's own random keys. Thus, the simulator can compute $m := (z_r, \pi_r) \leftarrow \mathsf{VRF.Eval}(\mathsf{otsk}^*_{r,t}, \mathsf{tran})$ and create an indistinguishable transcript by outputting (K, ρ, m).

If $\mathsf{otpk}_{r,t} = \perp$ and the real receiver sends a VRF on the semi-static key spk then they cannot obtain deniability. This is because a VRF is a deterministic signature and the real receiver has signed spk_r in σ_{spk_r}, implying knowledge of ssk_r. Since a VRF contains a NIZK, the simulator cannot create a valid VRF on spk_r without violating soundness/uniqueness since it does not know ssk_r.

If $(s, r) \in \mathcal{H} \times \mathcal{H}$: Sample random element $\mathsf{esk} \xleftarrow{\$} \mathbb{Z}_p$, sample random $K||\tau_{\mathsf{conf}}$ and compute ρ using esk and tran. Compute $m \leftarrow \mathsf{VRF.Eval}(\mathsf{otsk}^*_{r,t}, \mathsf{tran})$ since it knows the one-time secret key corresponding to the public key it replaced in the honest users' prekey bundles. Output (K, K, ρ, m).

Similar to above, we lose deniability immediately if the honest receiver uses ssk_r as the VRF key or $\mathcal{L}_{\mathsf{KDC}} = 1$.

$\mathsf{SimSt}_{\mathcal{H}}(\mathsf{isk}_r, \mathsf{st}_r)$: Parse $(D_{\mathsf{prek}}, D_{\rho\perp}) \leftarrow \mathsf{st}_r$ and delete the last unused one-time state, i.e., let t be the smallest index s.t. $D_{\mathsf{prek}}[t] \neq \perp$. If $t \leq L$, let $D_{\mathsf{prek}}[t] \leftarrow \perp$. Output the updated state st_r.
$\mathsf{SimSt}_{\mathcal{C}}(\mathsf{st}_{\mathsf{Sim}})$: For $u \in \mathcal{C}$ extract st_u from $\mathsf{st}_{\mathsf{Sim}}$, parse $(D_{\mathsf{prek}}, D_{\rho\perp}) \leftarrow \mathsf{st}_u$ and delete the ephemeral secrets associated to used prekeys, i.e., for $t \in [\mathsf{counter}_u]$, with $t \leq L$, let $D_{\mathsf{prek}}[t] \leftarrow \perp$. Output $(st_u)_{u\in\mathcal{C}}$.

F.4 Strong Deniability of X3DH+VRF

Lemma 6. *Assume the underlying group used by* X3DH+VRF *is a generic group of prime order p and* KDF *is modeled as a random oracle. The* X3DH + VRF *protocol is strongly global deniable with respect to KDC leakage bit $\mathcal{L}_{\mathsf{KDC}} = 0$ and leakage function $\mathcal{L}_{\mathsf{high}}$ if the one-time prekey bundles are never depleted; and it is not strongly local deniable for the receiver or strongly global deniable for honest senders and receivers otherwise. It is still strongly local deniable for honest senders for leakage function $\mathcal{L}_{\mathsf{high}}$ even if the last-resort prekey bundles are used.*

We will construct the simulator.

$\mathsf{SimPreK}$: Same as in Lemma 5 except we run $\mathcal{A}$ to get the prekey bundles for the accusers. Additionally we keep track of all GGM and random oracle queries in $\mathsf{aux}_{\mathsf{Sim}}$.
$\mathsf{SimTrans}_{\mathcal{A}}(\mathsf{isk}, \mathsf{st}_{\mathsf{Sim}}, (s, r, t))$: Here isk is either isk_s,isk_r or $\perp$ depending on $s \in \mathcal{C},\ r \in \mathcal{C}$ or $(s, r) \in \mathcal{H} \times \mathcal{H}$, respectively.

If $(s, r) \in \mathcal{H} \times \mathcal{C}$: First retrieve the prekey bundle $(\mathsf{spk}_r, \sigma_{\mathsf{spk}_r}, \mathsf{otpk}_{r,t}) \leftarrow \mathsf{prek}_{r,t}$ of the malicious receiver, sample $\mathsf{esk} \xleftarrow{\$} \mathbb{Z}_p$ and query the GGM labeling oracle to receive $\Xi(\mathsf{esk})$.
Check if spk_r is a Good_L label using $\mathsf{aux}_{\mathsf{Sim}} \in \mathsf{st}_{\mathsf{Sim}}$. If so, compute the unique signed prekey secret $\mathsf{ssk}_r \in \mathbb{Z}_p$ such that $\Xi(\mathsf{ssk}_r) = \mathsf{spk}_r$. Query the GGM labeling oracle on tuples $(\mathsf{ipk}'_s, \mathsf{ssk}_r), (\mathsf{ipk}'_r, \mathsf{esk}), (\mathsf{spk}_r, \mathsf{esk})$ and $(\mathsf{otpk}_{r,t}, \mathsf{esk})$ to obtain labels (s_1, s_2, s_3, s_4). Set

$$(\mathsf{ss}_1, \mathsf{ss}_2, \mathsf{ss}_3, \mathsf{ss}_4) = (s_1, s_2, s_3, s_4)$$

compute (K, ρ, temp) according to Send in Fig.17. Now internally run the malicious receiver algorithm $(m, \mathsf{st}_{\mathcal{A}}) \xleftarrow{\$} \mathcal{A}(\mathsf{isk}_r, \mathsf{st}_{\mathcal{A}}, \mathsf{ipk}_s, t, \rho)$. Run $\mathsf{sas}_r \leftarrow \mathsf{SenderReceive}(\mathsf{temp}, m)$ to see if the real sender would accept. If $(K \neq \perp)$ (i.e. the honest sender accepts) then output (K', ρ, m) else output $(\perp, \rho, m)$.
Otherwise, if spk_r is a Bad_L label then sample random $K||\tau_{\mathsf{conf}}$ from range of KDF, compute ρ as above. Sim can compute m as above if not out of one-time prekeys. If $\mathsf{otpk}_{r,t} = \perp$ then the malicious receiver will fail the check in $\mathsf{SenderReceive}$ since it cannot forge proofs on Bad_L spk_r. Output (K, ρ, m).

If $(s,r) \in \mathcal{C} \times \mathcal{H}$: First execute the malicious sender internally $(\rho, \mathsf{st}_{\mathcal{A}}) \xleftarrow{\$} \mathcal{A}(\mathsf{isk}_s, \mathsf{st}_{\mathcal{A}}, (s,r,t))$ using the accuser's state $\mathsf{st}_{\mathcal{A}} \in \mathsf{st}_{\mathsf{Sim}}$. The simulator updates $\mathsf{st}_{\mathsf{Sim}}$ by storing the updated $\mathsf{st}_{\mathcal{A}}$. Parse $(\mathsf{epk}, \tau_{\mathsf{conf}}, (z_s, \pi_s)) \leftarrow \rho$, check if $\mathcal{A}$ ever made a random oracle query on $(\mathsf{ss}, \mathsf{tran})$ where $\mathsf{tran} := \mathsf{ipk}_s || \mathsf{ipk}_r || \mathsf{prek}_{r,t} || \mathsf{epk}$ such that $K || \tau_{\mathsf{conf}} = \mathsf{KDF}(\mathsf{ss}, \mathsf{tran})$ using $\mathsf{aux}_{\mathsf{Sim}}$ and assert $\mathsf{Ver}(\mathsf{epk}, \mathsf{tran}, z_s, \pi_s)$. If not, output $(\perp, \rho, \perp, \mathsf{st}_{\mathsf{Sim}})$ since this is the case where the receiver's authentication check will fail and they will output $\perp$ for the key and also for m . Otherwise, retrieve the first such tuple (ss, K), compute $m \leftarrow \mathsf{VRF.Eval}(\mathsf{otsk}^*_{r,t}, \mathsf{tran})$ as above and output (K, ρ, m).

If the honest receiver runs out of prekey bundles and uses ssk_r to VRF then they lose all deniability similar to Lemma 5. If $\mathcal{L}_{\mathsf{KDC}} = 1$ then honest receiver has no deniability.

If $(s,r) \in \mathcal{H} \times \mathcal{H}$: First sample $\mathsf{esk} \xleftarrow{\$} \mathbb{Z}_p$ and query the GGM labeling oracle to receive $\mathsf{epk} := \Xi(\mathsf{esk})$. Then sample random $K || \tau_{\mathsf{conf}}$ from the range of KDF and compute ρ. Compute $m \leftarrow \mathsf{VRF.Eval}(\mathsf{otsk}^*_{r,t}, \mathsf{tran})$. Output (K, K, ρ, m). If honest receiver is out of one-time prekeys then a third-party accuser/simulator cannot compute m i.e. the VRF under spk_r and we lose global deniability.

$\mathsf{SimSt}_{\mathcal{H}}$ Parse $(D_{\mathsf{prek}}, D_{\rho\perp}) \leftarrow \mathsf{st}_r$ and delete the last unused one-time state, i.e., let t be the smallest index s.t. $D_{\mathsf{prek}}[t] \neq \perp$. If $t \leq L$, let $D_{\mathsf{prek}}[t] \leftarrow \perp$. Output the updated state st_r.

$\mathsf{SimSt}_{\mathcal{A}}(\mathsf{st}_{\mathsf{Sim}})$: Extract $\mathsf{st}_{\mathcal{A}}$ from $\mathsf{st}_{\mathsf{Sim}}$ and output $\mathsf{st}_{\mathcal{A}}$.

G Analysis of Deniability of X3DH+PVRF-SAS

We show that X3DH+PVRF-SAS (almost) retains the deniability properties of the underlying X3DH.

In Figures 18 and 19 we fully specify the algorithms of X3DH+PVRF-SAS as they are used in the deniability analysis of this scheme. Fig. 18 contains the IdKeyGen and PreKeyBundleGen algorithms, which are the same for X3DH, X3DH+VRF-SAS and X3DH+PVRF-SAS, and Fig. 19 contains the remainder of the X3DH+PVRF-SAS algorithms, with differences from plain X3DH marked.

Note that in X3DH+PVRF-SAS the SAS values sas_s and sas_r generated by honest parties are publicly computable as $\mathsf{sas} = z \oplus \mathsf{nonce}$ from the other parts of the key exchange transcript. We therefore omit $\mathsf{sas}_s, \mathsf{sas}_r$ throughout our deniability arguments, with the understanding that they can be trivially simulated by any algorithm that can simulate protocol messages ρ, m.

G.1 Deniability Against Honest-But-Curious Accusers

We begin by proving the deniability of X3DH+PVRF-SAS against passive accusers.

Proof (Theorem 5).

We first construct the simulator required for the deniability proof:

$\mathsf{IdKeyGen}(1^\kappa)$:	$\mathsf{PreKeyBundleGen}(\mathsf{isk}_u)$:
$\mathsf{isk}' \xleftarrow{\$} \mathbb{Z}_p,\ \mathsf{ipk}' := [\mathsf{isk}^*]G$	$(\mathsf{isk}'_u, \mathsf{sk}_u) \leftarrow \mathsf{isk}_u$
$(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{Sig.KeyGen}(1^\kappa)$	$D_{\mathsf{prek}}, D_{\rho\perp} := \varnothing$
return $(\mathsf{ipk} := (\mathsf{ipk}', \mathsf{vk}), \mathsf{isk} := (\mathsf{isk}', \mathsf{sk}))$	$\mathsf{ssk}_u \xleftarrow{\$} \mathbb{Z}_p,\ \mathsf{spk}_u := [\mathsf{ssk}]G$
	$\sigma_{\mathsf{spk}_u} \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}_u, \mathsf{spk}_u)$
	for $(t \in [L])$ **do**
	$\quad \mathsf{otsk}_{u,t} \xleftarrow{\$} \mathbb{Z}_p,\ \mathsf{otpk}_{u,t} := [\mathsf{otsk}_{u,t}]G$
	$\quad \mathsf{prek}_{u,t} := (\mathsf{spk}_u, \sigma_{\mathsf{spk}_u}, \mathsf{otpk}_{u,t})$
	$\quad D_{\mathsf{prek}}[t] \leftarrow (\mathsf{prek}_{u,t}, (\mathsf{ssk}_u, \mathsf{otsk}_{r,t}))$
	$\mathsf{prek}_{u,\perp} := (\mathsf{spk}_u, \sigma_{\mathsf{spk}_u}, \perp)$
	$D_{\mathsf{prek}}[L+1] \leftarrow (\mathsf{prek}_{u,\perp}, (\mathsf{ssk}, \perp))$
	return $(\mathbf{prek_u}, \mathsf{st}_u := (D_{\mathsf{prek}}, D_{\rho\perp}))$

Fig. 18: Algorithms IdKeyGen and PreKeyBundleGen of X3DH and X3DH+PVRF-SAS

- SimPreK: Honestly generate prekey bundles for accusers and do not overwrite the prekey bundles of honest users.
- $\mathsf{SimTrans}(\mathsf{isk}, \mathsf{st}_{\mathsf{Sim}}, (s, r, t))$: Here isk is either isk_s, isk_r, or $\perp$, depending on whether $s \in \mathcal{C}$, $r \in \mathcal{C}$, or $(s, r) \in \mathcal{H} \times \mathcal{H}$, respectively.
  - If $(s, r) \in \mathcal{H} \times \mathcal{C}$: First sample $\mathsf{esk} \xleftarrow{\$} \mathbb{Z}_p$ and set $\mathsf{epk} := [\mathsf{esk}]G$. Then extract the signed prekey ssk_r from $\mathsf{st}_r \in \mathsf{st}_{\mathsf{Sim}}$ and compute

    $$(\mathsf{ss}_1, \mathsf{ss}_2, \mathsf{ss}_3, \mathsf{ss}_4) := \left([\mathsf{ssk}_r]\mathsf{ipk}'_s, [\mathsf{esk}]\mathsf{ipk}'_r, [\mathsf{esk}]\mathsf{spk}_r, [\mathsf{esk}]\mathsf{otpk}_{r,t}\right)$$

    compute the rest of Send as in Fig. 19 to get (K, ρ, temp). Retrieve st_r, the accuser state, from $\mathsf{st}_{\mathsf{Sim}}$ and compute $(K', m, \mathsf{st}'_r) \leftarrow \mathsf{Receive}(\mathsf{isk}_r, \mathsf{st}_r, \mathsf{ipk}_s, t, \rho)$. Output (K, ρ, m).
  - If $(s, r) \in \mathcal{C} \times \mathcal{H}$: Run $(K, \rho, \mathsf{temp}, \mathsf{st}_s) \leftarrow \mathsf{Send}(\mathsf{isk}_s, \mathsf{ipk}_r, \mathsf{prek}_{r,t})$ using the accusing sender's long-term secrets.
    Retrieve vt and vts from ρ and temp respectively. Construct input tran to PECVRF as the transcript of the key exchange and run $m \leftarrow \mathsf{FakeEval}(\mathsf{otpk}_{r,t}, vt, vts, \mathsf{tran})$ and output (K, ρ, m).
    If $\mathsf{otpk}_{r,t} = \perp$ then the simulator can do the same for spk_r. However, if $\mathsf{leak} = \mathsf{high}$ then this simulation will fail because D can compute $w' := [\mathsf{ssk}]H$ and cross check with the transcript.
  - If $(s, r) \in \mathcal{H} \times \mathcal{H}$: Sample random element $\mathsf{esk} \xleftarrow{\$} \mathbb{Z}_p$, sample random $K || \tau_{\mathsf{conf}}$ and compute ρ and tran. Compute m similarly as above. Output (K, K, ρ, m).

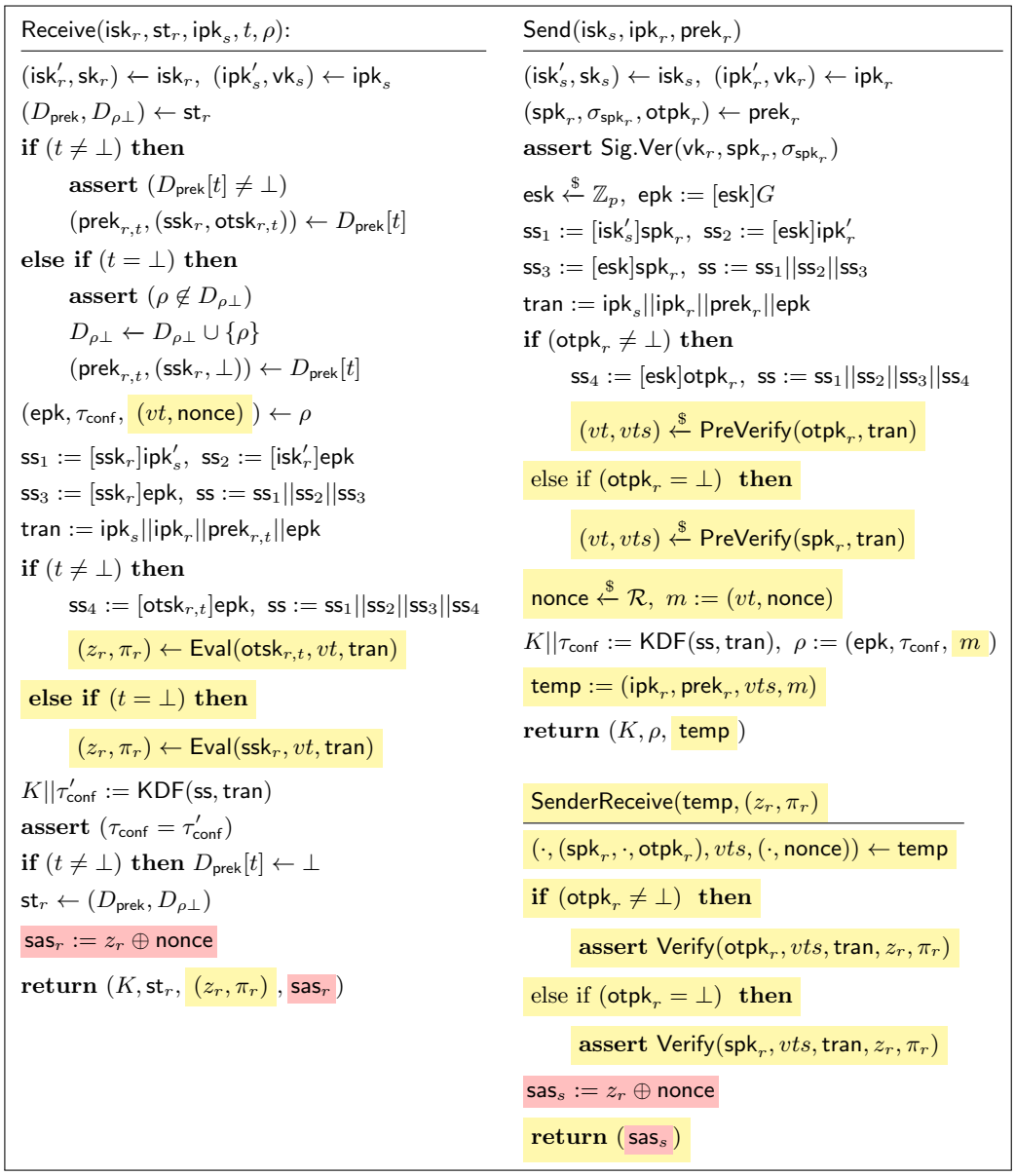

Fig. 19: Protocol X3DH+PVRF-SAS, with differences from X3DH marked in yellow and SAS outputs marked in pink. X3DH+PVRF-SAS algorithms IdKeyGen and PreKeyBundleGen are exactly as in X3DH, see Figure 18.

- $\mathsf{SimSt}_{\mathcal{H}}(\mathsf{isk}_r, \mathsf{st}_r)$: Parse $(D_{\mathsf{prek}}, D_{\rho\perp}) \leftarrow \mathsf{st}_r$ and delete the last unused one-time state, i.e., let t be the smallest index s.t. $D_{\mathsf{prek}}[t] \neq \perp$. If $t \leq L$, let $D_{\mathsf{prek}}[t] \leftarrow \perp$. Output the updated state st_r.
- $\mathsf{SimSt}_{\mathcal{C}}(\mathsf{st}_{\mathsf{Sim}})$: For $u \in \mathcal{C}$ extract st_u from $\mathsf{st}_{\mathsf{Sim}}$, parse $(D_{\mathsf{prek}}, D_{\rho\perp}) \leftarrow \mathsf{st}_u$ and delete the ephemeral secrets associated to used prekeys, i.e., for $t \in [\mathsf{counter}_u]$, with $t \leq L$, let $D_{\mathsf{prek}}[t] \leftarrow \perp$. Output $(st_u)_{u \in \mathcal{C}}$.

We can analyze local deniability for the case when one-time prekeys do not run out first. We have disclosure function $\mathcal{D}_{\mathsf{high}}$, leakage function $\mathcal{L}_{\mathsf{high}}$ and KDC leakage bit $\mathcal{L}_{\mathsf{KDC}} = 1$. By correctness of X3DH+PVRF-SAS since honest-but-curious accusers compute their PECVRF honestly SenderReceive will always verify and output a sas_s.

If $(s, r) \in \mathcal{H} \times \mathcal{C}$ notice that $\mathsf{ss}_2, \mathsf{ss}_3$ and ss_4 are generated exactly as the real world. The simulated $\mathsf{ss}_1 = [\mathsf{ssk}_r]\mathsf{ipk}'_s$ has the same distribution as the real world $\mathsf{ss}_1 = [\mathsf{isk}'_s]\mathsf{spk}_r$. Additionally, $m := (vt, \mathsf{nonce})$ has the same distribution as the real world since it can be computed publicly(using $\mathsf{otpk}_{r,t}$ and tran). Since we can use the accusing receiver's long-term secret, the simulator runs $\mathsf{Receive}(\mathsf{isk}_r, st_r, \mathsf{ipk}_s, t, \rho)$ on identically distributed inputs and generates $(K', \mathsf{st}_r, (z_r, \pi_r))$. Since the simulator can run the code a real receiver would run, this output is distributed exactly as the real world. Hence, the output of SimTrans is identical to the real world output from D's view, even given $\mathsf{disc}_{\mathsf{D}}$ and $\mathsf{leak}_{\mathsf{D}}$.

If $(s, r) \in \mathcal{C} \times \mathcal{H}$ then ρ can be computed exactly as in the real world by SimTrans by simply running $(K, \rho, \mathsf{temp}) \leftarrow \mathsf{Send}(\mathsf{isk}_s, \mathsf{ipk}_r, \mathsf{prek}_r)$. Since simulator cannot modify the prekey bundle it must compute the receiver's by running $(vt, vts) \leftarrow \mathsf{PreVerify}(\mathsf{otpk}_r, x)$ and then running $(z_r, \pi_r) \leftarrow \mathsf{FakeEval}(\mathsf{otpk}_r, vt, vts, x)$. The simulator can then run $\mathsf{SenderReceive}(\mathsf{temp}, (z_r, \pi_r))$ to get sas_r which will be distributed identically by correctness. The main difference from D's view is that simulator samples a random group element w while the real world receiver would compute it as $[\mathsf{otsk}_{r,t}]H$. We can build a reduction to PECVRF deniability to compute D's advantage.

If we consider the interaction D has with the attack oracle when $\mathsf{mode} = \mathsf{real}$ as Game_1 and the interaction it has with the attack oracle when $\mathsf{mode} = \mathsf{sim}$ as $\mathsf{Game}_{\mathsf{final}}$ then the global deniability game in Fig. [4,5] measure's D's advantage in distinguishing which Game_i for $i \in \{1, \mathsf{final}\}$ it is in.

Consider a series of hybrid games from $\mathsf{Game}_{n,0}$ to $\mathsf{Game}_{0,L}$ where n is the number of honest parties and L is the number of one-time prekeys generated by PreKeyBundleGen. The reduction between $\mathsf{Game}_{n,0}$ to $\mathsf{Game}_{n,1}$ embeds the PECVRF public key from the game for PVRF deniability, Fig. 2, into $\mathsf{otpk}_{n,1}$. $\mathsf{Game}_{n,0}$ runs the algorithm for $\mathsf{mode} = \mathsf{real}$(corresponding to $b = 0$ for PVRF deniability) entirely while $\mathsf{Game}_{n,1}$ runs the simulator for $\mathsf{otpk}_{n,1}$(corresponding to $b = 1$ for PVRF deniability) but the $\mathsf{mode} = \mathsf{real}$ algorithm for $\mathsf{otpk}_{n,i}$ where $i \geq 2$. If for any of these reductions D does not call the attack oracle on (s, r) such that the embedded $\mathsf{otpk}_{r,t}$ is not used (i.e. the corresponding secret must now be leaked for honest users since $\mathsf{leak} = \mathsf{high}$) the reduction aborts. This leaves D's view unchanged in either case. We can make this hybrid argument

over all parties so that for the reduction between $\mathsf{Game}_{i,j}$ and $\mathsf{Game}_{i,j+1}$ all honest receivers $r > i$ are simulated while all honest receivers $r < i$ run the real algorithm.

$$\Delta_{\mathsf{D}}(\mathsf{Game}_{n,0}, \mathsf{Game}_{0,L}) \leq n \cdot L \cdot \mathsf{Adv}^{\mathsf{Den}}_{\mathsf{D},\mathsf{PECVRF}}$$

Local receiver deniability if one-time prekeys run out. If the last-resort prekey bundle is used we can embed the challenge into spk_r. If leakage is restried to $\mathsf{leak} = \mathsf{med}$ then a similar line of reasoning can be applied to simulate key exchanges made using the last-resort key bundle resuling in the same bound.

If $(s, r) \in \mathcal{H} \times \mathcal{H}$ and the one-time prekey bundles do not run out we always have a one-time prekey $\mathsf{otpk}_{r,t}$ for $t \in [L]$ and the sender always computes the session key K by including ss_4 in the KDF. Disclosure and leakage are high. The crucial observation is that once an honest receiver runs $\mathsf{Receive}(\mathsf{isk}_r, \mathsf{st}_r, \mathsf{ipk}_s, t, \rho)$ the one-time prekey secret otsk_t is deleted from its state st_r. Specifically, D does not have $\mathsf{otsk}_{r,t}$ nor esk to compute $\mathsf{ss}_4 = [\mathsf{esk}]\mathsf{otpk}_{r,t} = [\mathsf{otsk}_{r,t}]\mathsf{epk}$ after the session is established.

Consider Game_1 which is the global deniability game for $\mathsf{mode} = \mathsf{real}$ in Fig. [4,5]. Consider a modified game Game_2 for the case when $(s, r) \in \mathcal{H} \times \mathcal{H}$ which runs the Send algorithm identically but in the $\mathsf{Receive}$ algorithm instead of using the correct secret key $\mathsf{otsk}_{r,t}$ and running Eval it computes $m \leftarrow \mathsf{FakeEval}(\mathsf{otpk}_{r,t}, vt, vts, \mathsf{tran})$ using the verification token secret it sampled earlier. $\mathsf{SenderReceive}$ does not need to be modified.

Following a similar set of reductions as above we can replace all the w's with random group elements.

$$\Delta_{\mathsf{D}}(\mathsf{Game}_1, \mathsf{Game}_2) \leq n \cdot L \cdot \mathsf{Adv}^{\mathsf{Den}}_{\mathsf{D},\mathsf{PECVRF}}$$

Consider Game_3 which is identical to Game_2 except in Send instead of computing ss_4 sample a random group element and use it as ss_4 and in $\mathsf{Receive}$ skip the key confirmation tag check. Construct a similar reduction as above but instead embed a DDH challenge in $(\mathsf{otpk}_{r,t}, \mathsf{epk}, \mathsf{ss}_4)$ like in proof of Theorem 1. Using a hybrid argument as above we get

$$\Delta_{\mathsf{D}}(\mathsf{Game}_2, \mathsf{Game}_3) \leq n \cdot L \cdot \mathsf{Adv}_{\mathsf{DDH}}$$

Consider $\mathsf{Game}_{\mathsf{final}}$ which is the simulated world where the simulator picks a random element from the codomain of KDF. The distinguisher can only compute ss_1,ss_2 and ss_3 part of input to the KDF and ss_4 is a random group element in Game_3. Since $\mathsf{otsk}_{r,t}$ is deleted after protocol execution just like esk the distinguisher cannot distinguish the simulated key and confirmation tag from the ones computed in the real world unless it queries KDF on the input computed in Game_3. Since the random group element is hidden from D the only way it distinguishes between Game_3 and $\mathsf{Game}_{\mathsf{final}}$ is by querying KDF on this random group element(along with the rest of the corresponding input). If D makes q queries to the KDF it will have distinguishing advantage as follows

$$\Delta_{\mathsf{D}}(\mathsf{Game}_3, \mathsf{Game}_{\mathsf{final}}) \leq \frac{q}{p}$$

Summing up the incremental game changes we get D's advantage in distinguishing between $\mathsf{mode} = \mathsf{real}$ and $\mathsf{mode} = \mathsf{sim}$ for $(s, r) \in \mathcal{H} \times \mathcal{H}$.

$$\therefore \Delta_{\mathsf{D}}(\mathsf{Game}_1, \mathsf{Game}_{\mathsf{final}}) \leq n \cdot L \cdot \mathsf{Adv}^{\mathsf{Den}}_{\mathsf{D},\mathsf{PECVRF}} + \cdot n \cdot L \cdot \mathsf{Adv}_{\mathsf{DDH}} + \frac{q}{p}$$

Global deniability if one-time prekeys run out. The DDH challenge can be embedded into $(\mathsf{spk}_r, \mathsf{epk}, \mathsf{ss}_3)$ to get a simulator for $\mathsf{leak} = \mathsf{med}$ assuming DDH is hard and KDF is a random oracle. We can no longer embed the DDH challenge into ss_4 as $\mathsf{otpk}_{u,\perp} = \perp$. If leakage is restricted to $\mathsf{leak} = \mathsf{med}$ then a similar line of reasoning as above can be applied to simulate key exchanges made using the last-resort key bundle. The challenge can be embedded in $\mathsf{ss}_3 = [\mathsf{esk}]\mathsf{spk}_r = [\mathsf{ssk}_r]\mathsf{epk}$ instead. Note that the reduction must now create new signatures on these embedded values using the secret it knows. Following an almost identical argument as above we can prove that no distinguisher D can have non-negligible advantage in the local deniability game assuming DDH is hard and KDF is a random oracle resulting in the same bound.

G.2 Deniability Against Malicious Accusers

We next prove the deniability of X3DH+PVRF-SAS against malicious accusers.

Proof (Theorem 6).
First we construct the simulator.

- $\mathsf{SimPreK}_{\mathcal{A}}\left((\mathsf{ipk}_u)_{u\in\mathcal{N}}, (\mathsf{isk}_u)_{u\in\mathcal{C}}, (\mathbf{prek_u})_{u\in\mathcal{H}}\right)$: Same as in Theorem 5 except we run $\mathcal{A}$ to get the prekey bundles for the accusers. Output $((\mathbf{prek_u})_{u\in\mathcal{H}}, (\mathbf{prek_u})_{u\in\mathcal{C}}, \mathsf{st}_{\mathsf{Sim}})$ with $\mathsf{st}_{\mathsf{Sim}} = ((\mathsf{ipk}_u, \mathbf{prek_u})_{u\in\mathcal{N}}, (\mathbf{otsk^*_u})_{u\in\mathcal{H}}, \mathsf{aux}_{\mathsf{Sim}})$. Keep track of all AGM and random oracle queries in $\mathsf{aux}_{\mathsf{Sim}}$.
- $\mathsf{SimTrans}_{\mathcal{A}}(\mathsf{isk}, \mathsf{st}_{\mathsf{Sim}}, (s, r, t))$: Here isk is either isk_s,isk_r or $\perp$ depending on $s \in \mathcal{C},\ r \in \mathcal{C}$ or $(s, r) \in \mathcal{H} \times \mathcal{H}$, respectively.
  - If $(s, r) \in \mathcal{H} \times \mathcal{C}$: First retrieve the prekey bundle $(\mathsf{spk}_r, \sigma_{\mathsf{spk}_r}, \mathsf{otpk}_{r,t}) \leftarrow \mathsf{prek}_{r,t}$ of the malicious receiver, sample $\mathsf{esk} \xleftarrow{\$} \mathbb{Z}_p$ and compute $\mathsf{epk} := [\mathsf{esk}]G$.
    Check in the AGM records if spk_r was formed properly i.e. whether $\mathcal{A} + \mathsf{Sim}$ knows discrete log of spk_r to base G. If so, retrieve the corresponding secret ssk_r and compute ss using ssk_r for ss_1 and esk for the other three DH values. Compute (K, ρ, temp) according to Send in Fig. 19. Internally run the malicious accuser on this ρ to get $(m, \mathsf{st}_{\mathcal{A}}) \xleftarrow{\$} \mathcal{A}(\mathsf{isk}_r, \mathsf{st}_{\mathcal{A}}, \mathsf{ipk}_s, t, \rho)$. Run $\mathsf{sas}_r \leftarrow \mathsf{SenderReceive}(\mathsf{temp}, m)$. Output (K, ρ, m).
    This simulator can also be applied to when $\mathsf{otpk}_{r,t} \neq \perp$.
    If spk_r is not formed properly then sample random $K||\tau_{\mathsf{conf}}$ from range of KDF, compute ρ as above using sampled τ_{conf}. Run $\mathcal{A}$ on this input

to get $(m, \mathsf{st}_{\mathcal{A}}) \xleftarrow{\$} \mathcal{A}(\mathsf{isk}_r, \mathsf{st}_{\mathcal{A}}, \mathsf{ipk}_s, t, \rho)$. Since $\mathcal{A}$ does not know the DL corresponding to spk_r it cannot verify tags sent from honest senders. It also cannot compute verifying proofs under spk_r. Run honest sender's algorithm $\mathsf{sas}_s \leftarrow \mathsf{SenderReceive}(\mathsf{temp}, m)$. Because of the non-malleability of PECVRF(implied by $\mathcal{F}_{\mathsf{PVRF}}$) an honest sender would abort and output $\perp$ since $\mathcal{A}$ cannot forge proofs on new inputs on keys it does not know the secret to. Output $(\perp, \rho, m)$.

- If $(s, r) \in \mathcal{C} \times \mathcal{H}$: First execute the malicious sender internally $(\rho, \mathsf{st}_{\mathcal{A}}) \xleftarrow{\$} \mathcal{A}(\mathsf{isk}_s, \mathsf{st}_{\mathcal{A}}, (s, r, t))$ using the accuser's state $\mathsf{st}_{\mathcal{A}} \in \mathsf{st}_{\mathsf{Sim}}$. Parse $(\mathsf{epk}, \tau_{\mathsf{conf}}, m) \leftarrow \rho$ where $m \leftarrow \mathsf{FakeEval}_{\mathcal{A}}(\mathsf{otpk}_{r,t})$, check if $\mathcal{A}$ ever made a random oracle query on $(\mathsf{ss}, \mathsf{tran})$ where $\mathsf{tran} := \mathsf{ipk}_s||\mathsf{ipk}_r||\mathsf{prek}_{r,t}||\mathsf{epk}$ such that $K||\tau_{\mathsf{conf}} = \mathsf{KDF}(\mathsf{ss}, \mathsf{tran})$ using $\mathsf{aux}_{\mathsf{Sim}}$. In PECVRF the vt contains a NIZK so an honest receiver will output $\perp$ since $\mathcal{A}$ cannot fake proofs on badly formed vt without beating soundness for the NIZK proof system. Thus, we do not need to consider deniability for cases when vt is formed improperly. Output (K, ρ, m).
- If $(s, r) \in \mathcal{H} \times \mathcal{H}$: Same as Theorem 5.

- $\mathsf{SimSt}_{\mathcal{H}}$ Parse $(D_{\mathsf{prek}}, D_{\rho\perp}) \leftarrow \mathsf{st}_r$ and delete the last unused one-time state, i.e., let t be the smallest index s.t. $D_{\mathsf{prek}}[t] \neq \perp$. If $t \leq L$, let $D_{\mathsf{prek}}[t] \leftarrow \perp$. Output the updated state st_r.
- $\mathsf{SimSt}_{\mathcal{A}}(\mathsf{st}_{\mathsf{Sim}})$: Extract $\mathsf{st}_{\mathcal{A}}$ from $\mathsf{st}_{\mathsf{Sim}}$ and output $\mathsf{st}_{\mathcal{A}}$.

First we analyze deniability when one-time prekeys do not run out. Notice there are only three cases in which the simulator deviates from the algorithms executed in the real world. Throughout the proof of deniability we assume that the algorithms D_1 and $\mathcal{A}$ are a single entity which for convenience of notation we call $\mathcal{A}$.

When $(s, r) \in \mathcal{H} \times \mathcal{C}$ and spk_r is not formed properly since D does not know ssk_r or esk the DH value ss_3 remains hidden. Specifically, the output of KDF remains indistinguishable from random as long as $\mathcal{A}$ or D does not query KDF on the correct ss_3.

The simulator in AGM sees how spk_r is formed. Say $\mathsf{spk}_r = [x]G + \sum_i [x_i]G_i + \sum_j [y_j]H_j$ where $G_i := [t_i]G$ are group elements to which the simulator knows discrete log to base G and H_j are all the random group elements the algebraic adversary and simulator receive. These might be honestly generated public keys by the reduction or random oracle outputs. Consider a reduction to CDH which takes as input two random group elements (A, B). Consider Game_1 which is identical to the real world and Game_2 which is identical to Game_1 except on the first attack oracle query to $(s, r) \in \mathcal{H} \times \mathcal{C}$ if spk_r is formed badly then it runs the simulator. The reduction sets $\mathsf{epk} := A$ and sets each $H_j := [r_j]G + [r'_j]B$ where r_j and r'_j are random field elements. To have any advantage in distinguishing $\mathcal{A}$ must query KDF on the right input. When $\mathcal{A}$ queries KDF on $(\mathsf{ss}, \mathsf{tran})$ such that tran contains epk and spk_r check if $y_j = 0$ for all j. If so, compute DL of spk_r to base G as $[x + \sum_i x_i t_i]$ and compute ss_3 correctly. This can be considered part of the case when spk_r is formed correctly since faker algorithm Sim can

create an identical transcript. Else if $y_j \neq 0$ for any j in the coefficients of H_j in spk_r then sample random $K||\tau_{\mathsf{conf}}$. Else compute ss_3 and input to KDF since $\mathsf{spk}_r = [x + \sum_j r_j]G + [x_i]G_i + [r'_j]B$. Therefore if $\mathsf{epk} := A$ then $\mathsf{ss}_3 = [x+\sum_j r_j]A+[x_i t_i]A+[\sum_j r'_j y_j]\mathsf{CDH}(A, B)$. Retrieve $\mathsf{CDH}(A, B)$ by subtracting the first two terms then multiplying the last term with the reciprocal of its coefficient and return to the reduction. Since the CDH challenge is embedded into every random group element if the simulator cannot create an identical transcript by extracting the spk_r secret then the reduction must solve CDH to have any distinguishing advantage on this attack oracle query. The reduction does not know which attack oracle query to embed the epk challenge in. However, since D could run the attack oracle on the same receiver for Q queries we consider a series of hybrids where in the reduction between Game_i and Game_{i+1} the reduction replaces the i-th attack oracle query's epk with A if $(s, r) \in \mathcal{C} \times \mathcal{H}$ and spk_r for this prekey bundle is not formed well. For all previous attack oracle queries $j < i$ if $(s, r) \in \mathcal{C} \times \mathcal{H}$ and spk_r for this prekey bundle is not formed well then those transcripts are simulated by sampling random $K||\tau_{\mathsf{conf}}$. Additionally, the reduction must guess the correct KDF query. If there are a total of q_{KDF} queries to KDF using a hybrid argument we get bound:

$$\Delta_{\mathsf{D}}(\mathsf{Game}_1, \mathsf{Game}_{Q+1}) \leq q_{\mathsf{KDF}} \cdot Q \cdot \mathsf{Adv}_{\mathsf{CDH}}$$

When $(s, r) \in \mathcal{C} \times \mathcal{H}$ and $\mathcal{A}$ made multiple queries on the same tran. $\mathcal{A}$ could query $\mathsf{ss} \neq \mathsf{ss}'$ such that $K||\tau_{\mathsf{conf}} := \mathsf{KDF}(\mathsf{ss}, \mathsf{tran})$ and $K'||\tau'_{\mathsf{conf}} := \mathsf{KDF}(\mathsf{ss}', \mathsf{tran})$. Since KDF is a random oracle $\tau_{\mathsf{conf}} \neq \tau'_{\mathsf{conf}}$ with overwhelming probability. Also, for any fixed τ_{conf} and tran it is still infeasible for $\mathcal{A}$ to find ss^* such that $K^*||\tau_{\mathsf{conf}} := \mathsf{KDF}(\mathsf{ss}^*, \mathsf{tran})$ for arbitrary K^*. Thus, if the honest receiver accepts then the malicious sender must have queried $(\mathsf{ss}, \mathsf{tran})$ such that $K||\tau_{\mathsf{conf}} := \mathsf{KDF}(\mathsf{ss}, \mathsf{tran})$. Additionally, a similar reduction to DDH and PVRF (strong) deniability as in Theorem 5 will give us similar bounds but here we call $\mathsf{FakeEval}_{\mathcal{A}}(\mathsf{otpk}_r)$.

When $(s, r) \in \mathcal{H} \times \mathcal{H}$ we can use a similar reduction as in Theorem 5 to get the same bounds on deniability in this case.