Key-schedule Security for the TLS 1.3 Standard

Chris Brzuska[1], Antoine Delignat-Lavaud[2], Christoph Egger[3], Cédric Fournet[2], Konrad Kohbrok[1], and Markulf Kohlweiss[4]

[1] Aalto University, Finland {chris.brzuska,konrad.kohbrok}@aalto.fi
[2] Microsoft Research Cambridge, UK {fournet,antdl}@microsoft.com
[3] Friedrich-Alexander University Erlangen, Germany christoph.egger@fau.de
[4] Edinburg University, UK mkohlwei@ed.ac.uk

Abstract. We analyze the security of the TLS 1.3 key establishment protocol, as specified at the end of its rigorous standardization process. We define a core key-schedule and reduce its security to concrete assumptions against an adversary that controls client and server configurations and adaptively chooses some of their keys. Our model supports all key derivations featured in the standard, including its negotiated modes and algorithms that combine an optional Diffie-Hellman exchange for forward secrecy with optional pre-shared keys supplied by the application or recursively established in prior sessions. We show that the output keys are secure as soon as *any* of their input key materials are. Our compositional, code-based proof makes use of state separation to yield concrete reductions despite the complexity of the key schedule. We also discuss (late) changes to the standard that would improve its robustness and simplify its analysis.

1 Analyzing the TLS 1.3 Handshake

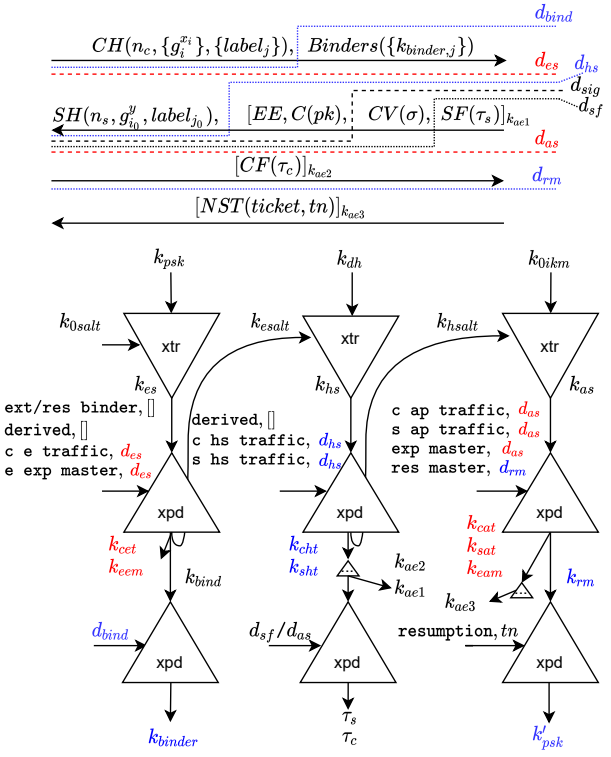
Transport Layer Security (TLS) is the most widely used authenticated secure channel protocol on the Internet, protecting the communications of billions of users. Previous versions of TLS have suffered from many attacks against weaknesses in their design, including legacy algorithms (e.g. FREAK for export RSA [11], LogJam [3] for export Diffie-Hellman, WeakDH for ill-chosen groups, and exploits against Mantin biases of RC4 [25]); the RSA key encapsulation (e.g. the ROBOT [22] variant of Bleichenbacher's PKCS1 padding oracle); the fragile MAC-encode-encrypt construction leading to many variants of Vaudenay's padding oracles against CBC cipher suites (e.g. BEAST, Lucky13 [4]); the weak signature over nonces allowing protocol version downgrades (e.g. DROWN [6] and POODLE); the key exchange (e.g. FREAK [54] and 3SHAKE [15]); the hash transcript (e.g. SLOTH [18]); and other negotiated parameters [13]. TLS 1.3 intends both to fix the weaknesses of previous versions and to improve their performance, notably by lowering the latency of connection establishment from two roundtrips down to one, or even zero when resuming a connection.

Historically, the IETF process to adopt a standard involves an open consortium of contributors mostly coming from industry, with a bias towards early implementers. The TLS working group at the IETF acknowledged that this process puts too much emphasis on deployment and implementation concerns, and

tends to address security issues reactively [55]. Instead, it decided to address security upfront by welcoming feedback from various cryptographic efforts, including symbolic [32,31] and computational protocol models [35,36,51], both on paper and implemented in tools such as Tamarin or CryptoVerif. Early drafts of TLS 1.3 also drew much inspiration from Krawczyk's OPTLS protocol [50], which comes with a detailed security proof, although later versions diverged from it (in particular in the design of resumption). This proactive approach has certainly improved the overall design of TLS 1.3, and uncovered flaws along its 28 intermediate drafts. However, many of these efforts are incomplete (focusing, e.g., on fixed protocol configurations) or do not account for the final version published in RFC 8446. Since final adoption, further questions have been raised about pre-shared keys, potential reflection attacks [38], and difficulties in separating resumption PSKs (produced internally by the key exchange) from external ones installed by the application. In short: we still miss provable security for the new Internet standard.

TLS can be decomposed into sub-protocols: the *record layer* manages the multiplexing, fragmentation, padding and encryption of data into packets (also called *records*) from three separate streams of handshake, alert, and application data. Incoming handshake messages are passed to the *handshake* sub-protocol, which in turn produces fresh record keys and outgoing handshake messages. Taking advantage of this well-understood modularity, other protocols re-use the TLS 1.3 handshake with different record layers: for instance, DTLS 1.3 is a variant based on UDP datagrams instead of TCP streams, while the IETF version of QUIC replaces the record layer with a much extended transport [44], adding features such as dynamic application streams and fine-grained flow control. Detailed security proofs for the TLS 1.3 record layer have been proposed by Patton et al. [56] (extending the work of Fischlin et al. [41] on stream-based channels), Badertscher et al. [7], and Bhargavan et al. [14], who also provide a verified reference implementation. Therefore, we defer to these works for the record layer, and focus on the handshake protocol.

TLS 1.3 Handshake and Key Schedule The top of Fig. 1 gives an abstract view of the protocol message flow. In the client hello (CH) message, the client sends a nonce n_c, its Diffie-Hellman (DH) share g^x, a PSK *label* and a *binder* value (the role of which we explain in our key schedule security model). As a means of negotiation, the client may offer shares for different groups and different PSK options (thus the indices i, j in $g_i^{x_i}, label_j, binder_j$). The server communicates its choice of the DH group and the PSK when sending the server hello (SH) message which contains the server nonce n_s, its share $g_{i_0}^y$ (including the group description) and the label $label_{j_0}$ of the chosen PSK. The remaining messages consist of encrypted extensions (EE), server certificate and signature $(\mathrm{C}(pk), \mathrm{CV}(\sigma))$, key confirmation messages in the forms of messages authentication codes (MACs) τ_s and τ_c computed over the transcript, and a *ticket* which is used on the client side to store a resumption key (later referred to as *resumption PSK*) derived from the key material of the current key exchange session.



Parent name(s):		
n:	n_1	n_2
es:	$0salt$	psk
eem:	es	$\perp$
cet:	es	$\perp$
$bind$:	es	$\perp$
$binder$:	$bind$	$\perp$
hs:	$esalt$	dh
sht:	hs	$\perp$
cht:	hs	$\perp$
$hsalt$:	hs	$\perp$
as:	$hsalt$	$0ikm$
rm:	as	$\perp$
cat:	as	$\perp$
sat:	as	$\perp$
eam:	as	$\perp$
psk:	rm	$\perp$

$$
\begin{aligned}
k_{es} &\leftarrow \mathsf{xtr}(k_{psk}, k_{0salt}) \\
k_{bind} &\leftarrow \mathsf{xpd}(k_{es}, \texttt{ext/res binder}, []) \\
k_{binder} &\leftarrow \mathsf{xpd}(k_{bind}, [], d_{bind}) \\
k_{cet} &\leftarrow \mathsf{xpd}(k_{es}, \texttt{c e traffic}, d_{es}) \\
k_{eem} &\leftarrow \mathsf{xpd}(k_{es}, \texttt{e exp master}, d_{es}) \\
k_{esalt} &\leftarrow \mathsf{xpd}(k_{es}, \texttt{derived}, []) \\
k_{hs} &\leftarrow \mathsf{xtr}(k_{esalt}, k_{dh}) \\
k_{cht} &\leftarrow \mathsf{xpd}(k_{hs}, \texttt{c hs traffic}, d_{hs}) \\
k_{sht} &\leftarrow \mathsf{xpd}(k_{hs}, \texttt{s hs traffic}, d_{hs}) \\
k_{hsalt} &\leftarrow \mathsf{xpd}(k_{hs}, \texttt{derived}, [])
\end{aligned}
\qquad
\begin{aligned}
k_{as} &\leftarrow \mathsf{xtr}(k_{hsalt}, k_{0ikm}) \\
k_{cat} &\leftarrow \mathsf{xpd}(k_{as}, \texttt{c ap traffic}, d_{as}) \\
k_{sat} &\leftarrow \mathsf{xpd}(k_{as}, \texttt{s ap traffic}, d_{as}) \\
k_{eam} &\leftarrow \mathsf{xpd}(k_{as}, \texttt{exp master}, d_{as}) \\
k_{rm} &\leftarrow \mathsf{xpd}(k_{as}, \texttt{res master}, d_{rm}) \\
k'_{psk} &\leftarrow \mathsf{xpd}(k_{rm}, \texttt{resumption}, tn) \\
k_{psk},\, k_{0ikm}, k_{0salt} &\leftarrow 0^{\mathsf{len}(alg)} \\
\tau_{s/c} &\leftarrow \mathsf{deriveFin}(k_{sht}/k_{cht}, d_{sf}/d_{as}) \\
\sigma &\leftarrow \mathsf{sign}(sk, d_{sig}) \\
k_{ae\{1,2,3\}} &\leftarrow \text{TLS record layer derivations}
\end{aligned}
$$

Fig. 1: TLS 1.3 Handshake Protocol (top) and Key Schedule (bottom). Keys $k\_$ depend on handshake transcript digests $d\_$ and are output by the key schedule at each phase of the protocol. After the negotiation of algorithms, shares and pre-shared keys in the client hello (CH) and server hello (SH), the handshake traffic secrets (k_{cht}, k_{sht}) are output, and used to derive the handshake encryption keys (k_{ae1}, k_{ae2}) and the HMAC keys for the server finished (SF) and client finished (CF) key confirmation messages. In typical configuration, the server is authenticated by sending its certificate $C(pk)$ and signing the digest in the certificate verify $CV(\sigma)$ message. See Fig. 11a for a formal description of the key schedule and Fig. 18 for how it integrates into the TLS 1.3 handshake. Our model focuses on the core key schedule that produces traffic secrets. Note that we use $\mathsf{xpd}(\cdot, [], \cdot)$ as an alias for $\mathsf{hmac}(\cdot, \cdot)$. This encoding is sound since xpd is not used with an empty label, otherwise. The encoding is convenient, since it allows us to model k_{binder} as a key—note that hmac and xpd both provide pseudorandomness and collision-resistance, justifying their uniform treatment (in fact, xpd is based on hmac). We color digests and keys in alternating red and blue to clarify digest-key dependency.

The *key schedule* is the core part of the handshake that performs all key computations. It takes as main input PSK and DH key materials and, at each phase of the handshake, it derives keys, e.g., to encrypt client early traffic (k_{cet}), to compute the binder value (k_{binder}), to encrypt server handshake traffic (k_{sht}) and to encrypt client handshake traffic (k_{cht}). In Fig. 1, each column of triangles (from left to right) respectively describes the *early* phase, the *handshake* phase and the *application* phase of the schedule.

The key schedule thus consists of a collection of xtr an xpd operations, organized in a graph. Each of the operations takes as input a *chaining* key and/or new key material, (k_{psk} in the xtr in the early phase and k_{dh} in the xtr in the in the handshake phase), together with the latest digest and auxiliary inputs such as a resumption status r and a ticket nonce tn. In particular, on the path from an input key to an output key, one xpd operation includes the latest digest of the protocol messages d_{bind}, d_{es}, d_{hs}, d_{as}, and d_{rm} which causes the output key to depend on k_{binder} and the Diffie-Hellman shares indirectly via the transcript hash. The computation of the DH secret k_{dh} from DH shares and exponents is also part of our modeling, despite happening outside the xtr and xpd functions. From each chaining key, at least one output key and/or new chaining key for the next invocation will be derived via xtr and xpd.

Our TLS key schedule model thus outputs seven keys: k_{cet}, k_{eem}, k_{binder}, k_{cht}, k_{sht}, k_{cat}, k_{sat}, k_{eam}. They constitute a natural boundary for our model, inasmuch as all other TLS keys and IVs are further derived from them in a transcript-independent manner.

Extracting a Key Schedule Model for the Handshake We intend to capture all aspects of key-exchange security that relate to the creation and derivation of keys that are output by TLS sessions. Our resulting *key-schedule security model* is an partial specification of key exchange security for TLS that elides the content and flow of protocol messages, how the state of each participant advances, and the mapping between cryptographic identities (such as DH shares and PSKs) and participant identities (such as certificates, server names, PSK identifiers, etc.). In this model, the adversary can (i) register chosen dishonest application PSKs and dishonest DH shares, (ii) instruct the game to sample honest application PSKs and honest DH shares, and (iii) trigger key derivations. This overapproximation of the adversaries capabilities allows for a simpler model (in a similar way as adversarially chosen plaintexts simplify models of encryption). The adversary of the key schedule can freely inject and derive corrupt keys across many resumptions and include (essentially) arbitrary additional parameters such as transcripts. These capabilities go beyond practically possible attacks against TLS. Informally, the model ensures that all output keys are *pairwise distinct* and, moreover, if *honest* key material was used at any point in the derivation, then the derived keys are *pseudorandom* and *pseudo-independent*, i.e., they look like independently drawn, uniformly random strings.

While the key schedule model captures the cryptographically interesting security properties of the key exchange, it does not capture other aspects such as authentication or key confirmation. For instance, who actually owns a key is a

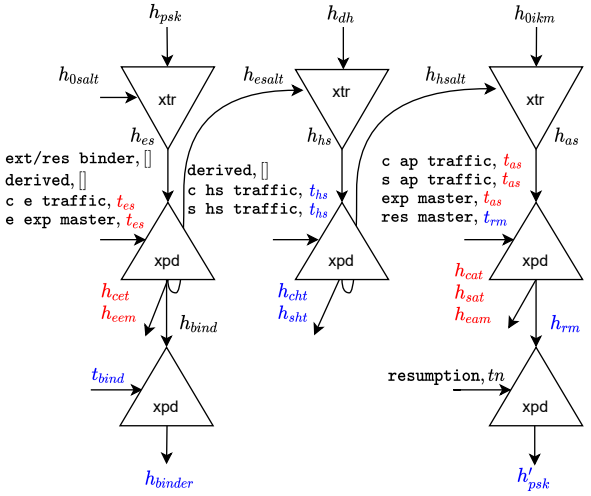
protocol property, and the corresponding party identities are left implicit in the key schedule. Of course, if the identity can be unambiguously parsed from the plaintext handshake transcript, then agreement on session keys (which implicitly authenticates this transcript in its derivation) implies in particular that the peers agree on the server's certificate. We argue that these derived properties are more suitable for tool-assisted security modeling, inasmuch as their proofs are conceptually simple (e.g. perfect reduction to code-based assumptions, or symbolic Dolev-Yao models) but involve many details of the protocol. For instance, tools such as Tamarin are excellent at detecting missing binding between identities and protocols keys (see, e.g., the post-handshake client authentication attack based on transcript collisions with PSKs reported by Cremers et al. [31] in an earlier draft). Thus, we propose to use computational (handwritten) analysis such as ours to precisely capture the pseudorandomness and independence properties achieved by the key schedule (and pinpoint the underlying assumptions) for all configurations supported by the standard, and then rely on verification tools to develop a comprehensive model of the handshake. This paper focuses on the TLS 1.3 key schedule model, and leaves a mechanically-verified model of the handshake based on key-schedule security as future work.

Key Handles and Collision Attacks In key exchange models, cryptographic *session identifiers* are used to match client and server instances that agree on the session key and possibly other parameters such as participant identities and the session transcript (see [28] for a discussion on agreement and [52,13] for a discussion on session identifiers in key exchange). The mapping from session identifiers to keys should be *injective* with overwhelming probability, i.e., two protocol instances that derive the same key should have the same identifier. In this sense, it is useful to think of a session identifier as containing all the relevant parts of the communication transcript that influence the key.

Our key schedule model uses *handles* as the analogue of session identifiers in protocols. The handles of output keys are closely related to session identifiers in the key exchange. More systematically, we introduce handles for all internal, intermediate keys (such as k_{es} and k_{esalt}). We construct handles recursively, based on (the handles of) their key material and the additional inputs implicitly authenticated in their key derivation (label, transcript, ticket nonce, as detailed in Fig. 2). To this end, we define injective handle constructor functions $\mathsf{xtr}\langle\cdot\rangle$, $\mathsf{xpd}\langle\cdot\rangle$, $\mathsf{dh}\langle\cdot\rangle$ as the modelling counterpart of concrete key derivation functions. Fig. 2 sums up this notation and shows how handles are chained through the TLS key schedule. See Section 4.2 for the formal definition of handles.

Handles act as *administrative* identifiers used by the adversary to request operations on keys whose values are secret. Handles are also crucial in expressing *uniqueness*: for output keys, we show that, with overwhelming probability, there is a one-to-one mapping between handles and values. Finally, as illustrated above, uniqueness provides a general basis for authenticating all key-exchange parameters included in the handles.

Internally, uniqueness of the output keys depends on uniqueness of their input key materials, which is problematic both for Diffie-Hellman secrets and for



$n = psk$:
$h_{psk} \leftarrow \mathsf{appPSK}\langle ctr, alg\rangle$
or $h_{psk} \leftarrow \mathsf{noPSK}\langle alg\rangle$
$n = dh$:
$X_{i_0} \leftarrow g_{i_0}^{x_{i_0}}; Y_{i_0} \leftarrow g_{i_0}^{y}$
$h_{dh} \leftarrow \mathsf{dh}\langle\mathsf{sort}(X_{i_0}, Y_{i_0})\rangle$
or $h_{dh} \leftarrow \mathsf{noDH}$
$(n, salt, ikm) \in \{(es, 0salt, psk),$
$(hs, esalt, dh), (as, hsalt, 0ikm\}$:
$h_n = \mathsf{xtr}\langle n, h_{salt}, h_{ikm}\rangle$
$(n, m) \in \{(bind, es), (esalt, es),$
$(cet, es), (eem, es), (hsalt, hs),$
$(cht, hs), (sht, hs), (rm, as),$
$(cat, as), (sat, as), (eam, as),$
$(psk, rm)\}$:
$h_n \leftarrow \mathsf{xpd}\langle n, \mathtt{label}_n, h_m, t\rangle$
$n = binder$:
$h_{binder} \leftarrow \mathsf{xpd}\langle n, [], h_{bind}, t\rangle$

Fig. 2: The rules for constructing handles h_n mirror the concrete key derivations of Fig. 1, but carry more information than the keys they represent (such as hash and group algorithms), to model cryptographic agility. The administrative handles of application PSKs consist of a counter *ctr* and the hash algorithm.

PSKs (both application-provisioned and derived for resumption). For DH shares (g^x, g^y), we construct the handles for the secret g^{xy} as $\mathsf{sort}(g^x, g^y)$. However, (g^{xr}, g^y) and (g^x, g^{yr}) clearly lead to the same key, so we can ensure neither uniqueness nor pseudorandomness at this level—for instance, if we combine such (dishonest) DH secrets with an honest PSK, the result should be pseudorandom and *look independent* while it would be identical. The TLS key schedule prevents collision attacks emerging from the collision of (g^{xr}, g^y) and (g^x, g^{yr}) by including the pair of shares into the authenticated transcript. Accordingly, our key schedule model restricts the adversary to use only transcripts whose Diffie-Hellman shares match those included in the handle.

A similar issue appears when the adversary registers a dishonest *resumption* PSK as a dishonest *application* PSK. TLS prevents key collision attacks emerging from such PSK collisions by including a different label (`ext` or `res`) into the derivation of the binder value, which appears in the handshake transcript (see Fig. 1) and is in turn hashed into the derivation of output keys. Thus, our model also restricts the adversary to use only transcripts with valid PSK binders.

As explained below, our model fully captures the potential use of different groups and hash algorithms at each phase of the key schedule (an important feature of TLS) which creates additional challenges to achieving uniqueness.

Security Assumptions We assume that the hash function for computing the transcript hash as well as the key derivation functions xtr and xpd are *collision-*

resistant[5]. For the early phase, we rely on the Extract-then-Expand paradigm (denoted xtr-then-xpd in our language) being a key derivation function (KDF) as by the HKDF standard [48], assuming that xtr is a secure extractor and xpd is a pseudorandom function (PRF). By the design of TLS 1.3, we assume that the PSKs are generated *independently* of the salt (which is set to 0 in the early phase) and that xtr is a secure extractor even with a 0 salt.

For the handshake and the application phase, we additionally rely on xtr being a PRF when *keyed with the salt*. The use of this assumption on xtr was suggested in the OPTLS work by Krawczyk and Wee [50] and is backed by the implementations of xtr via HMAC, which was shown to be an (arbitrary input-length) PRF [8,43], assuming the dual PRF security of its (fixed-length) compression function under (weak) related-key attacks. Additionally, we rely on the PRF-properties of xpd which, in turn, is also backed by the implementation of xpd via HMAC. Finally, we rely on the pseudorandomness of the MAC algorithm used to derive binder values (see Fig. 1), which is, once again, covered by the assumption that HMAC is a PRF.

An assumption which our work brought to light is the need to rely on the *salted* Oracle Diffie-Hellman (SODH) for Diffie-Hellman, i.e., if a Diffie-Hellman (DH) secret is generated from two random, honest shares, then applying xtr to the DH secret with several, adversarially chosen salts should still provide (computationally) independent and pseudorandom keys (see Section 3.3). In practice, such different salts can emerge from disagreement between server and client about the PSK to use since the early salt *esalt* changes when the PSK changes (see Fig. 1).

Cryptographic Agility The assumptions we make need to hold even if several hash algorithms are used in parallel and some of them may be broken. We model broken algorithms by allowing the adversary to choose the keys for the respective primitives, a strengthening of the fact that the adversary *knows* the keys derived by broken algorithms. Such *agile* assumptions on cryptographic primitives are typically quite strong. Interestingly, as we show, we can reduce several of the assumptions to non-agile assumptions for the specific case of TLS 1.3. This is possible because all hash functions currently supported in TLS 1.3 have a different output length and thus, the resulting keys can be distinguished due to the hash functions' separate domains. The only case where we need to make an agile base assumption (rather than prove agile security from non-agile security) is SODH. This is required because honest clients and servers may use the same Diffie-Hellman secret with different hash algorithms.

1.1 Proof Methodology and Outline

Our analysis of the TLS 1.3 Key Schedule involves three techniques of independent interest: (i) the fine-grained decomposition of games to manage their shared state which leads, essentially, to a graph-based induction (or, formally

[5] The TLS 1.3 standard acknowledges this collision resistance requirement: `https://www.rfc-editor.org/rfc/rfc8446.html#appendix-E.1.1`.

hybrid) proof, (ii) the abstraction of this technique into a general graph-based key schedule theory, and (iii) the careful management of collisions in the system. We elaborate on each of these techniques in turn.

Game Modularity In preparation for the proof, we decompose the pseudocode of the key schedule game into separate code packages, each with their own private state, that call one another, such that their composition (by code inlining) is perfectly equivalent to the initial, monolithic game.

This technique allows us to cut out part of the game (formally defined as a connected sub-graph of the directed, acyclic call graph of packages) and consider the rest of the game as a reduction, i.e., part of the adversary against the subgame. In particular, we define security assumptions so that we can perform game-hops (reductions) by matching and rewriting sub-graphs. The underlying mechanism is known as *state-separating proofs* (SSP [27]); we give a short introduction in Section 2.

Abstraction Since TLS derives a large number of keys, enumerating all reduction steps is a mechanical procedure which is not a useful task in itself. Instead of enumerating all steps, we provide a more general approach, namely, we define the class of all *TLS-like* key schedules and show that the graph-based approach applies to them. This way, we classify the different derivation steps in the key schedule according to the properties on which we rely. This classification is of conceptually independent interest, since other key schedules might rely on similar concepts for separating output keys.

Separating keys: Collision-resistance via Aborts and Mappings An important concept in our proof are *separation points*. Namely, the TLS 1.3 key schedule aims to provide distinct output keys, but does not include the resumption level as a label into the key derivation, thus making the separation between application keys (resumption level 0) and resumption keys (resumption level > 0) intricate. A separation point is necessary between the use of a pre-shared key and a derived output key in order to ensure separation between application keys and resumption keys. This separation point either includes a bit—as is the case for the computation of the *binder*—or it includes the binder into its arguments, inheriting the separatedness of the binder. Separation points provide a similar properties for DH keys.

Before a separation point, we have *redundant* keys, i.e., keys with different handles but identical values. Redundant keys complicate the use of assumption: A PRF, on the same inputs yields the same output and thus, when applying the security of a PRF based on the first input, we need to ensure that the second input—which in the case of xtr can also be a key—does not repeat.

While we use collision-resistance on separated keys, we need to use a separate mechanism on non-separated keys since, as just discussed, they might have inherent redundancy. We thus introduce a mapping that is *non-injective* on non-separated keys (which are intermediate keys in the key schedule), and we prove that the mapping is *injective* on separated keys and, in particular, injective on output keys which the adversary has access to. Therefore, proving pseudorandomness and uniqueness of the output keys using *internal* handles implies the

pseudorandomness and uniqueness of the output keys using *external* handles. To prove this rigorously, we define a relation over the state of two games, one using internal and one using external keys, and we show via induction over the oracle calls that the relation holds throughout and that if the relation on the state of the two games holds, then their input-output-behaviour is identical. We here import a proof technique from the area of formal verification, applying *invariants* for functional equivalence of pseudo-code in a paper proof.

We now outline the proof of the pseudorandomness and uniqueness of the output keys of the key schedule using internal handles (Theorem D.1). For details, see Appendix D.

Proof Outline

1. We idealize the transcript hash such that different transcripts always lead to different hash values.
2. We introduce a check for collisions that aborts the game instead of storing the same value for two *different dishonest handles*. The proof for this hop follows from collision-resistance of the derivation primitives, and we also use induction over the construction rules for keys and handles (see Fig. 2). The induction start relies on the uniqueness of dishonest application PSKs (the adversary is only allowed to store each application PSK value once)[6], the separation between application and resumption PSKs and uniqueness of dishonest DH secrets as guaranteed by the mapping.
3. We then introduce a check for collisions on *honest early salts* that aborts the game with a special *win* symbol instead of storing a value for a honest early salts that would collide with a prior early salt, honest or dishonest (Lemma D.4). The adversary technically wins when it causes a collision; Lemma D.7 removes the symbol from the game and bounds the loss.
4. We replace the handshake salt values (*hsalt*) and the output keys derived from honest DH secrets with fresh uniformly random values (Lemma D.5).
5. Relying on the pseudorandomness properties of the xtr/xpd derivations at every level, we replace all honest keys with uniformly random keys of the same length. (Lemma D.6). The proof follows from a hybrid argument both over the depth of the resumption tree and the derivation steps at each resumption level.
6. We introduce a check for collisions that aborts the game instead of storing the same value for different honest and dishonest output keys and binders. (Lemma D.7). For collisions between honest and dishonest keys, we rely on the pre-image resistance of the xtr and xpd computation: it is hard to find a pre-image of a uniformly random output value (as honest keys are already idealized). We bound any collisions between two honest keys by a birthday bound. An analogous argument allows us to analyze and then remove the special *win* aborts for collisions with honest early salts introduced in Lemma D.4.

   After this step, output keys and binders are unique and, if honest, random.

[6] PSK handles are administrative. As they are not shared with anyone else, an adversary with multiple handles for the same PSK value would only 'confuse' itself.

1.2 Related Work

In this paper, we model simultaneous protocol executions using different ciphersuites based on the same keying material. We capture the flexible way in which TLS negotiatiates ciphersuites and security parameters by considering keys tagged with algorithms. See [37,13] for a discussion on negotiation and [17] for a model and proof of TLS 1.2 that considers agility.

The following discussion focuses on attacker capabilities and security guarantees, and glosses over the exact encoding into security games and the use of multiple keys and stages.

Early drafts of TLS 1.3 were based on the OPTLS protocol by Krawczyk and Wee [50], and used semi-static Diffie-Hellman shares instead of pre-shared keys. Kohlweiss et al. [46] investigate a modified variant of the `draft-06` TLS 1.3 handshake in the constructive cryptography framework [53].

Dowling et. al. [35,36] present a multi-stage security model of `draft-05` and `draft-10` using variants of the Bellare-Rogaway key exchange models. Their multi-stage model considers `psk_ke`, `dh_ke`, and `psk_dhe_ke` modes in isolation. Li et al. [51] adapt the multi-stage security model to also capture the recursive nature of the TLS 1.3 key schedule, by accounting for the re-use of resumption secrets between different modes (`psk_ke`, `psk_dhe_ke`, and the now removed semi-static share 0-RTT). All of these works proved various subsets of the TLS standard secure.

Cremers et al. [32,31] investigate the security of `draft-10` and `draft-21`, using the automated Tamarin prover (in the symbolic model). Their work investigates the proposed post-handshake client authentication and finds an attack that exploited a missing binding between PSKs and transcripts that led to the addition of binders to the standard. To our knowledge ours is the first reduction proof that models the additional security afforded by binder values.

Bhargavan et al. [12] also model TLS 1.3, decomposed into 3 separate pieces: `dh_ke` 1-RTT handshake, the 0-RTT handshake, and the record protocol. They verify these models using both ProVerif [21] and CryptoVerif [19]. A limitation of their model is the informal way in which the separate guarantees for the three components are combined to justify the overall security of the protocol.

Blanchet [20] introduces a new proof modularization framework in CryptoVerif, which bears significant similarities with the state-separating proof framework [27] that our work is based on. The work also updates some of the model from `draft-18` to `draft-28`; however, the model still assumes that all pre-shared keys are derived from resumption secrets and does not capture adaptively-created dishonest application PSKs, or the security of PSK binders.

Many other works focus on analysing certain properties of the TLS 1.3 handshake protocol. For instance, Arfaou et. al. [5] specifically analyse the privacy of the TLS 1.3 `psk_ke`, `dh_ke`, and `psk_dhe_ke` handshakes. Fischlin et. al. [42] analyse the `draft-06` TLS 1.3 handshake, and show that its modes achieve key confirmation in isolation. Fischlin et. al. [40] considers replay attacks against various drafts of TLS 1.3 0-RTT handshakes such as `draft-14`'s `psk_ke` mode,

similarly considering versions and modes in isolation. Other relevant papers on TLS handshake analysis are [49,38,30].

The idea of analyzing a key schedule (rather than a key exchange protocol) is conceptually similar to the SIGMA-I pattern of Krawczyk [47] and Krawczyk and Wee [50]. In the simpler setting considered in these works, the relation between key exchange and key schedule can be performed by hand which is a conceptual validation of the approach taken in the current paper.

Recent work also looked at the tightness of TLS 1.3 security proofs [34,33]. Besides natural birthday bounds for collision resistance, our reductions avoid the common quadratic loss in the number of sessions. We remark however, that tightness was not the principal focus of our analysis.

In follow-up work, Brzuska et al. [26] analyze the key schedule and tree key encapsulation mechanism (TreeKEM) of Draft 11 of the messaging layer security standard using the SSP methodology in a similar style. Dupressoir et al. [39] show how SSP proofs can be used for a formalization in EasyCrypt and Abate et al. [1] formalize SSPs in Coq as the *SSProve* tool.

2 State-Separating Proofs

Our technical approach is based on state-separating proofs (SSP) [27], a variant of the code-based game-playing proofs of Bellare and Rogaway [10]. Following their approach, we define games and oracles by their pseudo-code. To enable proofs on large constructions, we further decompose games into packages, each with their own interface and private state. We refer to [27] for a formal presentation of SSPs. We now give an overview of a short proof (without code), illustrated on key expansion (xpd). The next sections provide examples of pseudocode.

Definition 2.1 (Packages). *A* package *is a collection of named* oracles*, whose code share access to private local state, and may in turn call oracles defined in other packages. Its* output interface *consists of the names of its oracles. Its* input interface *consists of the names of the oracles they call.*

Packages can be composed by connecting their input and output interfaces. A game *is a package whose oracles are fully defined: its input interface is empty.*

Consider a function $\mathtt{xpd}(k, x)$ that takes a (fixed-length) key and a label and returns a (fixed-length) bitstring—TLS uses such functions (with an additional digest argument) to create independent keys for each label. We may model its security as indistinguishability between two games with a single $\mathsf{XPD}(x)$ oracle: a real game $\mathtt{Gxpd}^0$ whose oracle returns $\mathtt{xpd}(k, x)$ where k is a private variable initialized at random, and an ideal game $\mathtt{Gxpd}^1$ whose oracle returns instead random results, using a private table indexed by x to memorize these results. Fig. 3 introduces our visual notation for these packages. By convention, we prefix

Fig. 3: $\mathtt{Gxpd}^b$ game

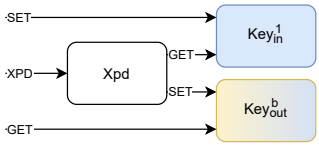


Fig. 4: Composed definition for game $\mathtt{Gxpd}^b$.

packages with G to indicate their use as security games. The adversary interacts with one of the games $\mathtt{Gxpd}^b$ and aims to determine the challenge bit b. We model this adversary as a package with input interface $\{\mathsf{XPD}\}$ and with a oracle that runs its code and returns its guess. We write $\mathcal{A} \to \mathtt{Gxpd}^b$ for its composition with our games, and define its advantage as $\mathsf{Adv}(\mathcal{A}, \mathtt{Gxpd}^b) := \left|\Pr[1 = \mathcal{A} \to \mathtt{Gxpd}^0] - \Pr[1 = \mathcal{A} \to \mathtt{Gxpd}^1]\right|$.

In protocols, the key used for key expansion may be produced by some other package, and the derived keys it produces may in turn be used in later steps of the key schedule. To enable compositional proofs, we thus split our Gxpd game into three packages: a core Xpd package and two Key packages that hold the keys it consumes and produces. As detailed in Section 2, each of these key packages holds multiple key instances, indexed by handles ranged over by h. The Key packages have oracles $\mathsf{SET}(h, k)$ and $\mathsf{GET}(h)$ to set and get their keys, respectively. Depending on their idealization bit b, they either store and return the concrete value passed to SET (if $b = 0$) or sample, store, and return a fresh random value (if $b = 1$). The core XPD package is now stateless: its $\mathsf{XPD}(h, x)$ oracle calls $\mathsf{GET}(h)$ to retrieve the input key k, then calls $\mathsf{SET}((h, x), \mathtt{xpd}(k, x))$ to store the derived key in a table at index (h, x). Fig. 4 on the right defines the resulting, analogous definition of our XPD game. Its top key package is always ideal (since expansion is keyed with a random key) whereas the bottom key package is parameterized by the game's challenge bit b. For the rest of the example, we let $\mathtt{Gxpd}^b$ refer to this composed package.

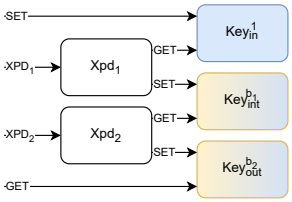


Fig. 5: Composed game $\mathtt{Gxxpd}^{b_1,b_2}$

We now illustrate SSPs by reducing the security of two-step expansion (simplified from TLS) to the security of its two steps. The first step derives an intermediate key, e.g. a traffic secret or a resumption master secret from which further output keys can be derived. Both steps are modeled using the XPD package described above, with different domains for their keys, and different names for their oracles (with two oracles $\mathsf{XPD}_1(h, x)$ and $\mathsf{XPD}_2(h, x)$ for the 2-step expansion).

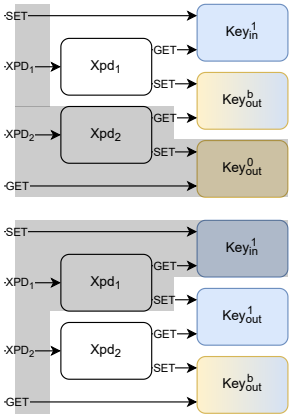


Fig. 6: Reducing to the $\mathtt{Gxpd}_1$ assumption, reduction $\mathcal{R}_1$ in gray (top). Reducing to the $\mathtt{Gxpd}_2$ assumption, reduction $\mathcal{R}_2$ in gray (bottom).

Fig. 5 gives a hybrid definition of the two-step expansion game parametrized by their two bits b_1 and b_2. The proof that output keys are uniformly random proceeds in two steps. We first idealize $\mathtt{Gxpd}_1$, switching b_1 from 0 to 1 (Fig. 6). We then idealize $\mathtt{Gxpd}_2$, now keyed with a random intermediary key, switching b_2 from 0 to 1 (Fig. 6). By combining the two steps, we obtain:

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gxxpd}^{0,0}, \mathtt{Gxxpd}^{1,1}) \leq \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_1, \mathtt{Gxpd}_1^b) + \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_2, \mathtt{Gxpd}_2^b).$$

Notation We use capital letters for oracle names and write $\mathtt{P}.\mathsf{ORACLE}(x)$ to refer to oracle ORACLE with argument x in package $\mathtt{P}$. In code, we use lowercase names for auxiliary functions. We write T, K,... for tables and sets; $\perp$ for errors; $\mathbf{h}$ for a sequence of h; $x \leftarrow e$ for assigning the result of e to x; and $x \leftarrow\!\!\$\, K$ for sampling x uniformly at random in K.

We write **throw** *event* for the command that immediately returns *event* to the adversary, skipping any further oracle computation or state change. Afterwards, all oracles are considered *silenced* and immediately return $\perp$, a behaviour introduced by Rogaway and Zhang in [57]. We use this command to model collisions within indistinguishibility games. We write **assert** c as an abbreviation for **if** c : **throw** $\perp$ and use it in oracles to restrict their use by the adversary.

We write $\mathsf{Adv}(\mathcal{A}, \mathtt{G}_0, \mathtt{G}_1)$, or just $\mathsf{Adv}(\mathcal{A}, \mathtt{G}^b)$, for the advantage of $\mathcal{A}$ distinguishing between two games $\mathtt{G}_0$ and $\mathtt{G}_1$, i.e., $\left|\Pr\left[\mathcal{A} \to \mathtt{G}^0\right] - \Pr\left[\mathcal{A} \to \mathtt{G}^1\right]\right|$.

3 Security Assumptions

We now define the assumptions that our protocol analysis relies on: collision resistance, (dual) pseudorandomness for the PRFs (xtr) and KDFs (xpd), and salted oracle Diffie-Hellman.

3.1 Collision-Resistance

Fig. 7 defines the collision-resistance game $\mathtt{Gcr}^{\mathsf{f}\text{-}alg,b}$ for a given function $\mathsf{f}\text{-}alg$. The HASH oracle takes as input a text t from the domain of $\mathsf{f}\text{-}alg$ and returns its digest d. If that text t has not been queried before, the digest is stored in table H at index t. In the ideal game ($b = 1$), the oracle first checks whether d already occurs in H, and if so, throws an abort. Hence, the adversary can distinguish between the real and the ideal game if and only if it can submit two different texts with the same digest. Our definition generalizes to n-ary functions by letting the text t be the tuple of their arguments. TLS 1.3 supports an extensible set of hash algorithms, currently defined as $\mathcal{H} = \{\mathtt{sha256}, \mathtt{sha384}, \mathtt{sha512}\}$ (see FIPS 180-2) and it employs these algorithms in three functionalities; hence, we rely on collision-resistance of the functions $\mathsf{hash}\text{-}alg$, $\mathsf{xtr}\text{-}alg$, and $\mathsf{xpd}\text{-}alg$ for every $alg \in \mathcal{H}$. We write $\mathsf{len}(alg)$ for the output length of alg.

$\underline{\underline{\mathtt{Gcr}^{\mathsf{f}\text{-}alg,b}}}$
$\mathsf{HASH}(t)$

assert $t \in dom(\mathsf{f}\text{-}alg)$
$d \leftarrow \mathsf{f}\text{-}alg(t)$
if $H[t] = \perp$:
  if $b \wedge d \in range(H)$:
    throw *abort*
  $H[t] \leftarrow d$
return d

Fig. 7: $\mathtt{Gcr}^{\mathsf{f}\text{-}alg,b}$ code.

3.2 Pseudorandomness

Fig. 8 defines pseudorandomness for a given two-input functions $\mathsf{f}\text{-}alg$. Since we may need to apply the same assumption twice, we let $\mathsf{f}\text{-}alg^\dagger$ be $\mathsf{f}\text{-}alg$ with its arguments swapped, i.e. $\mathsf{f}\text{-}alg^\dagger(k, x) := \mathsf{f}\text{-}alg(x, k)$.

We now describe each of the games. In the real pseudorandomness game $\mathtt{Gpr}^{\mathsf{f}\text{-}alg,0}$, the adversary submits values x to the oracle EVAL. Upon the first call, a random key k is sampled, then the oracle returns $\mathsf{f}\text{-}alg$ evaluated on k and

x. In the ideal game $\texttt{Gpr}^{\mathsf{f}\text{-}alg,1}$, the oracles returns instead truly random values, memorized in table T. Our main theorem assumes pseudorandomness of $\mathsf{xtr}\text{-}alg$ with $alg \in \mathcal{H}$, formalized in game $\texttt{Gpr}^{\mathsf{f}\text{-}alg,b}$ with $\mathsf{f}\text{-}alg = \mathsf{xtr}\text{-}alg$, dual pseudorandomness $\texttt{Gpr}^{\mathsf{f}\text{-}alg,b}$ with $\mathsf{f}\text{-}alg = \mathsf{xtr}^{\dagger}\text{-}alg$ and pseudorandomness of $\mathsf{xpd}\text{-}alg$ with $alg \in \mathcal{H}$, formalized in $\texttt{Gpr}^{\mathsf{f}\text{-}alg,b}$ with $\mathsf{f}\text{-}alg = \mathsf{xpd}\text{-}alg$.

$\texttt{Gpr}^{\mathsf{f}\text{-}alg,0}$

$\mathsf{EVAL}(x)$

if $k = \bot$:
  $k \leftarrow\!\!\$\ \{0,1\}^{\mathsf{len}(\mathsf{f}\text{-}alg)}$
return $\mathsf{f}\text{-}alg(k, x)$

$\texttt{Gpr}^{\mathsf{f}\text{-}alg,1}.\mathsf{EVAL}(x)$

if $T[x] = \bot$:
  $T[x] \leftarrow\!\!\$\ \{0,1\}^{\mathsf{len}(\mathsf{f}\text{-}alg)}$
return $T[x]$

Fig. 8: Oracles of the pseudorandomness game ($\texttt{Gpr}^{\mathsf{f}\text{-}alg,b}$).

3.3 Salted Oracle Diffie-Hellman (SODH)

The decisional Diffie-Hellman assumption (DDH) states that, given two honestly generated group elements $X = g^x$ and $Y = g^y$, the Diffie-Hellman secret g^{xy} is indistinguishable from a uniformly random group element [23]. To account for extraction and active adversaries, the oracle Diffie-Hellman assumption (ODH) keeps g^{xy} private and states that the values extracted from an honest Diffie-Hellman secret are pseudorandom, even if the adversary is given an oracle where it can submit values Z and sees, e.g., $\mathsf{hash}(Z^x)$ [2,24]. The variant of oracle Diffie-Hellman that we consider (SODH) additionally accounts for *salting*: the adversary also submits a salt value s and sees $\mathsf{xtr}(s, Z^x)$. Our assumption states that, even in this case, $\mathsf{xtr}(s, g^{xy})$ is pseudorandom.

TLS 1.3 supports an extensible set $\mathcal{G}$ of group descriptions, currently $\{\texttt{secp256r1}, \texttt{secp384r1}, \texttt{secp521r1}, \texttt{x25519}, \texttt{x448}\}$. Accordingly, we tag all group elements $(X, g^x, Y, \dots)$ with some $grp \in \mathcal{G}$, and write, e.g., $\mathsf{grp}(X)$ to access their group description.

The $\texttt{Gsodh2}^{grp,b}$ game defined in Fig. 9 captures this property as follows. The adversary calls DHGEN to instruct the game to generate honest Diffie-Hellman shares; their private exponents are stored in table L. The adversary calls XTR to use them to extract secrets for some chosen key shares and salt; the oracle checks that these argument are well-formed and that at least the first share has been honestly generated. In the real game ($b = 0$), or if the second share is dishonest, the oracle then computes $\mathsf{xtr}(salt, Y^x)$ and returns it to the adversary. Otherwise ($b = 1$ and both shares are honest), the oracle samples a fresh

$\mathsf{DHGEN}()$

$g \leftarrow \mathsf{gen}(grp)$
$q \leftarrow \mathsf{ord}(g)$
$x \leftarrow\!\!\$\ Z_q$
$X \leftarrow g^x$
$L[X] \leftarrow x$
return X

$\mathsf{XTR}(X, Y, salt)$

assert $L[X] \neq \bot$
  $\wedge\ \mathsf{grp}(X) = \mathsf{grp}(Y) = grp$
  $\wedge\ \mathsf{alg}(salt) \in \mathcal{H}$
$alg \leftarrow \mathsf{alg}(salt)$
if $b \wedge L[Y] \neq \bot$:
  $h \leftarrow \mathsf{dh}\langle\mathsf{sort}(X, Y)\rangle$
  if $S[h, salt] = \bot$:
    $S[h, salt] \leftarrow\!\!\$\ \{0,1\}^{\mathsf{len}(alg)}$
  return $S[h, salt]$
return $\mathsf{xtr}\text{-}alg(salt, Y^{L[X]})$

Fig. 9: Oracles of the (monolithic) game $\texttt{Gsodh2}^{grp,b}$ for the group description grp, where $\mathsf{gen}(grp)$ returns the group generator, $\mathsf{ord}(g)$ the order of the generator and $\mathsf{sort}(X, Y)$ sorts shares X and Y lexicographically. $\mathsf{alg}(salt)$ returns the algorithm tag of the adversary's $salt$.

value and returns it to the adversary. For consistency between repeated queries, this value is memorized in table S indexed by sorted key shares and salt.

Contrary to the PRF-ODH assumptions of Brendel, Fischlin, Günther and Janson [24], our assumption treats client and server key shares symmetrically. It is also agile in its hash algorithms (the algorithm tag of *salt* is chosen by the adversary), modelling the fact that honest TLS clients and servers may use the same shares with different algorithms. On the other hand, it does not require agility in the group description (*grp* is a fixed parameter of the game), since the key shares exchanged by TLS always carry this parameter, encoded as a named group descriptor. Just as in [24], one can give a heuristic argument for the validity of the assumption, by proving it in the random oracle model under a computational Diffie-Hellman assumption.

SODH implies PRF-ODH. Besides, SODH allows the adversary to choose the salt based on *both* shares, whereas PRF-ODF only allows for dependency on one of the shares. This is required in TLS 1.3 as the salt used by the client depends on server messages that may be modified by the adversary. In contrast, prior analysis of TLS 1.3, e.g., [35], give guarantees only to clients that use the same salt as the server. (In [35] this is modeled by setting lost to true whenever $\mathsf{cid}_i \neq \mathsf{cid}'_i$. We conjecture their proofs would not go through if the contributive identifier cid_i only consisted of key shares.)

4 Key Schedule Syntax and Security

We reason about the TLS 1.3 key schedule in terms of its three elementary operations extract (xtr), expand (xpd) and computation of Diffie-Hellman secrets. This section first introduces an abstract key schedule syntax and refines it to capture TLS 1.3 as part of a bigger class of *TLS-like* key schedules. We then define key schedule security and state our theorem for TLS-like key schedules.

4.1 Key Schedule Syntax

Our formalization interprets the key schedule as a directed graph where nodes describe *key names* (this is different from the data-flow graph in Fig. 1 where nodes describe *operations* and from the call graphs of packages for SSP). Each node (name) n has 0, 1 or 2 ingoing edges, depending on whether it is a *base key* (no parents), a result of an xpd operation (1 parent) or a result of an xtr operation (2 parents). We capture this by the parent name function PrntN which maps each name to its (ordered) pair of parents (each of which can be $\perp$). In addition to the set of names N and the graph description (encoded as PrntN), a key schedule has a function Label which maps the name and a resumption bit to a derivation label. We conveniently model hmac operations by using xpd with *empty label* as an alias for hmac. By sound cryptographic practice, a key should be either used for xpd or for hmac, so if a node has an empty label, it is not allowed to have siblings. Similarly, xtr operations only yield a single child, and the multiple children of xpd operations are derived using distinct labels.

Definition 4.1 (Key Schedule Syntax). *A key schedule* $ks = (N, \mathsf{Label}, \mathsf{PrntN})$ *consists of a set of names* N *and two functions*

$$\begin{aligned} \mathsf{Label}: &\quad N \times \{0,1\} \to \{0,1\}^{96} \cup \{\bot\} \\ \mathsf{PrntN}: &\quad N \to (N \cup \bot) \times (N \cup \bot) \end{aligned}$$

with the previously described restrictions.

$$\begin{aligned} es &\mapsto 0salt, psk \\ hs &\mapsto esalt, dh \\ as &\mapsto hsalt, 0ikm \end{aligned}$$

Fig. 10: parent names for xtr operations

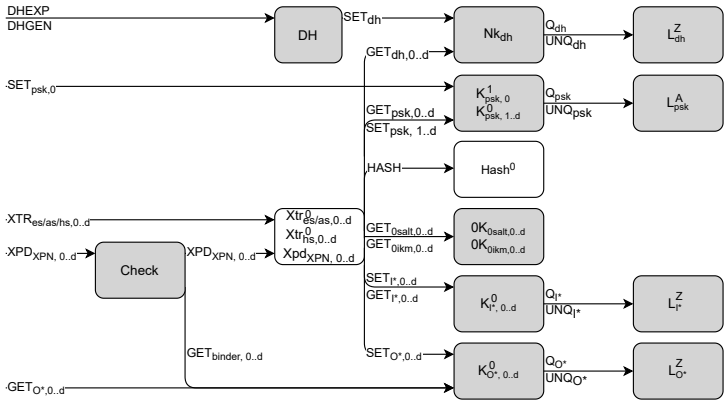
The table on the right side of Fig. 1 introduces the TLS key schedule in terms of the parent name function PrntN, which maps each key to up to two parent names. Stating and proving our theorem in terms of the concrete TLS key schedule requires listing and treating each xpd operation individually. Instead, we prove our theorem for all *TLS-like* key schedules (of which the TLS key schedule as described in Fig. 1 is an instance). We consider a key schedule as *TLS-like* if it aligns with TLS in terms of base keys and xtr operations and treats the *psk* name as the main root from which all keys except for the base keys can be reached. Moreover, a TLS-like key schedule only has a single loop. This loop contains the *psk* and models resumptions.

Definition 4.2 (TLS-like Key Schedule Syntax). *A key schedule* $ks = (N, \mathsf{Label}, \mathsf{PrntN})$ *is* TLS-like *if its graph satisfies the above restrictions, its set of names* N *contains at least the names 0salt*, *psk*, *es*, *esalt*, *dh*, *hs*, *hsalt*, *0ikm*, *as*, *rm and the parent name function* PrntN *maps* $0salt$*, dh and 0ikm to* $(\bot, \bot)$*, maps es, hs and as according to Fig. 10, maps psk to* $(rm, \bot)$ *and each of the remaining names* n *to some pair* $(n_1, \bot)$ *with* $n_1 \neq \bot$*.*

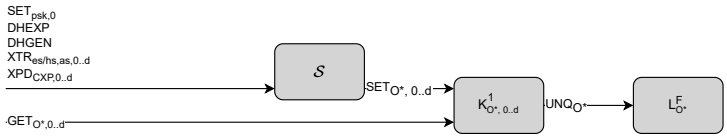
4.2 Key Handles

Complex derivation steps make it crucial to maintain administrative *handles* in the game state, both for internal bookkeeping and security modeling as well as for communication with the adversary. For honest keys, i.e., those sampled by the model, computed based on honest Diffie-Hellman secrets and derived via xtr and xpd from honest base keys, the adversary is not given the key itself, but rather a handle which the adversary can use to instruct the game to perform further computations on the key. We define handles as nested data records which each keep track of a step used to compute the associated key. We have several base handles for PSKs and DH secrets as well as their dummy zero values for noDH and noPSK mode and base handles for a fixed *0salt* and fixed *0ikm* (left upper triangle and right upper triangle in Fig. 2, respectively).

$\mathsf{dh}\langle\mathsf{sort}(X,Y)\rangle$	Diffie-Hellman secret
$h = \mathsf{psk}\langle ctr, alg\rangle$	application PSK
noDH	fixed zero Diffie-Hellman secret
$\mathsf{noPSK}\langle alg\rangle$	fixed zero PSK
0salt	fixed zero salt
$\mathsf{0ikm}\langle alg\rangle$	fixed zero initial key material (IKM)



(a) Real Game $\texttt{Gks}^0$



(b) Ideal Game $\texttt{Gks}^1(\mathcal{S})$

Fig. 11: Key schedule security games with internal keys I^* and output keys O^*. We write $\texttt{0K}_n$ as an abbreviation for $\texttt{Nk}_n \to \texttt{L}_n^Z$. We initialize K with 0 values as mentioned in Section 4.2 (See Fig. 23b in the Appendix for a formal discussion)

All other handles are then built from the base handles via inductively applying the following two constructors:

$$\mathsf{xtr}\langle \textit{name}, \textit{left parent handle}, \textit{right parent handle}\rangle.$$
$$\mathsf{xpd}\langle \textit{name}, \textit{label}, \textit{parent handle}, \textit{other arguments}\rangle.$$

For example, given a handle to the early master secret h_{es}, the handle to the client early transport secret (derived from the early master secret in one step) is defined as

$$h_{cet} = \mathsf{xpd}\langle cet, \texttt{c e traffic}, h_{es}, t_{es}\rangle$$

where t_{es} is the transcript of the protocol messages exchanged so far, and '`c e traffic`' is the constant byte string label prescribed in the RFC for this derivation step.

The handle always determines the named functionality (the index *cet*) and its algorithm descriptor, in case it is agile, written $\mathsf{alg}(h_{cet})$. We write $\mathsf{tag}_h(k)$ for key k tagged with this algorithm. For convenience, we write $\mathsf{len}(h_{cet})$ as an alias for $\mathsf{len}(\mathsf{alg}(h_{cet}))$. When we write h_{cet} we assume that this handle is of the above form. The handle indicates the actual key-derivation step, in this case

$$k_{cet} = \mathsf{xpd}(k_{es}, \texttt{c e traffic}, d_{es}).$$

Recall that, to support agility, our keys carry an algorithmic descriptor as metadata. In the example above, the derivation depends on the hash algorithm

$\mathtt{Xpd}_{n,\ell}$

Parameters

n : name
ℓ : level

$\mathsf{PrntN} : N \to (N_\perp \times N_\perp)$
$\mathsf{Labels} : N \times \{0,1\} \to \{0,1\}^{96}$

State

no state

$\mathsf{XPD}_{n,\ell}(h_1, r, args)$

$n_1, \_ \leftarrow \mathsf{PrntN}(n)$
$label \leftarrow \mathsf{Labels}(n, r)$
$h \leftarrow \mathsf{xpd}\langle n, label, h_1, args\rangle$
$(k_1, hon) \leftarrow \mathsf{GET}_{n_1,\ell}(h_1)$
if $n = psk$:
  $\ell \leftarrow \ell + 1$
  $k \leftarrow \mathsf{xpd}(k_1, (label, args))$
else
  $d \leftarrow \mathsf{HASH}(args)$
  $k \leftarrow \mathsf{xpd}(k_1, (label, d))$
$h \leftarrow \mathsf{SET}_{n,\ell}(h, hon, k)$
return h

$\mathtt{Xtr}^{b}_{n,\ell}$

Parameters

n : name
ℓ : level
b : bit
$\mathsf{PrntN} : N \to (N_\perp \times N_\perp)$
$\mathsf{Labels} : N \times \{0,1\} \to \{0,1\}^{96}$

State

no state

$\mathsf{XTR}_{n,\ell}(h_1, h_2)$

$n_1, n_2 \leftarrow \mathsf{PrntN}(n)$
if $\mathsf{alg}(h_1) \neq \perp \wedge \mathsf{alg}(h_2) \neq \perp$:
  assert $\mathsf{alg}(h_1) = \mathsf{alg}(h_2)$
$h \leftarrow \mathsf{xtr}\langle n, h_1, h_2\rangle$
$(k_1, hon_1) \leftarrow \mathsf{GET}_{n_1,\ell}(h_1)$
$(k_2, hon_2) \leftarrow \mathsf{GET}_{n_2,\ell}(h_2)$
$k \leftarrow \mathsf{xtr}(k_1, k_2)$
$hon \leftarrow hon_1 \vee hon_2$
if $b \wedge hon_2$:
  $k^\star \leftarrow\!\!\$\ \{0,1\}^{\mathsf{len}(k)}$
  $k \leftarrow \mathsf{tag}_{\mathsf{alg}(k)}(k^\star)$
$h \leftarrow \mathsf{SET}_{n,\ell}(h, hon, k)$
return h

DH

Parameters

G : set of groups
$\mathsf{ord} : G \to \mathbb{N}$

State

E : table

$\mathsf{DHGEN}(grp)$

assert $grp \in G$
$g \leftarrow \mathsf{gen}(grp)$
$x \leftarrow\!\!\$\ Z_{\mathsf{ord}(grp)}$
$X \leftarrow g^x$
$E[X] \leftarrow x$
return X

$\mathsf{DHEXP}(X, Y)$

assert $\mathsf{grp}(X) = \mathsf{grp}(Y)$
$h \leftarrow \mathsf{dh}\langle\mathsf{sort}(X, Y)\rangle$
$hon_X \leftarrow E[X] \neq \perp$
$hon_Y \leftarrow E[Y] \neq \perp$
assert $hon_X = 1$
$x \leftarrow E[X]; k \leftarrow Y^x$
$hon \leftarrow hon_X \wedge hon_Y$
$h \leftarrow \mathsf{SET}_{dh}(h, hon, k)$
return h

Fig. 12: Code for key derivation

for the xpd derivation step. The algorithm is defined by the handle h_{es} and implicitly passed to xpd as part of k_{es}.

The handle data also defines a *level*: the number of resumptions it records, counting from 0 and adding one for each node with a `resumption` label. We write $\mathsf{level}(h_{cet})$ for this level. Note that we use handle data also to communicate the handshake mode to the key schedule. A noDH Diffie-Hellman handle signals a `psk_ke` mode, while a $\mathsf{noPSK}\langle alg\rangle$ PSK handle signals a `dh_ke` mode. Similarly, we introduce handles $\mathsf{0ikm}\langle alg\rangle$ for the dummy key value $0^{\mathsf{len}(alg)}$ as well as 0salt for the 1-bit-long 0-key. This is because hmac (see Fig. 19) pads keys with zeroes up to their block length and thus, storing multiple zero values would introduce redundancy in the model without a correspondence in real-life.

4.3 Key Schedule Security Model

We aim to capture that the key schedule produces keys which are pseudorandom and unique. We define the security of the key schedule with the help of an ideal functionality which returns uniformly random and unique keys to the adversary. Security of a key schedule then demands that the real execution of the key schedule is indistinguishable from a simulated execution of the key schedule, performed by a simulator who does not have access to the honest keys provided to the adversary by the ideal functionality.

We describe the real execution of the key schedule as a game $\texttt{Gks}^0$, written in pseudocode. Following the SSP methodology outlined in Section 2, we split the pseudocode of the game $\texttt{Gks}^0$ into several packages, the functionality and meaning of each of which we describe in the subsequent subsections. Fig. 11a depicts the composed game $\texttt{Gks}^0$—recall that this graph is not merely an illustration, it is part of the formal definition of $\texttt{Gks}^0$.

Similarly, we describe an ideal game $\texttt{Gks}^1(\mathcal{S})$, parametrized by a simulator $\mathcal{S}$, see Fig. 11b. The $\texttt{K}^1_{O^*,0..d}$ and $\texttt{L}_{O^*}$ package which we define in Section 4.5 constitute the ideal functionality, namely, the $\texttt{K}^1_{O^*,0..d}$ package samples a uniformly random key for handles which correspond to honest keys with a name $n \in O^*$ and some level $0 \leq \ell \leq d$—our notion of honesty corresponds to what is referred to as *freshness* in the key exchange literature. We elaborate on honesty in Section 4.6. The $\texttt{L}_{O^*}$ package, in turn, ensures that each handle corresponds to a *different* key, modeling key uniqueness for both honest and dishonest keys.

The game $\texttt{Gks}^0$ exposes $\mathsf{SET}_{psk,0}$ and DHGEN oracles to allow the adversary to set honest Diffie-Hellman shares as well as (potentially dishonest) application PSKs. The adversary can then instruct the game to perform key derivations using Xtr and Xpd oracles. Finally, the output keys can be accessed by the GET oracle on the (real) key package $\texttt{K}^0_{O^*,0..d}$. Real key packages store concrete keys and do not rely on randomness.

In turn, in the ideal game (Fig. 11b), the $\texttt{K}^1_{O^*,0..d}$ packages answer the adversary's GET queries for output keys while the simulator is responsible for all other responses (and does not see the results returned to the adversary in a GET. The (ideal) key packages $\texttt{K}^1_{O^*,0..d}$ answer with a randomly sampled key (see code in Fig. 14) once the simulator decides a key should be available. This cleanly models the functionality of a key schedule, where after the protocol execution the participants have access to a shared, random key.

Definition 4.3 (Key Schedule Advantage). *For a key schedule* $ks = (N, \mathsf{Label}, \mathsf{PrntN})$*, a natural number* d*, a simulator* $\mathcal{S}$ *and an adversary* $\mathcal{A}$ *which makes queries for at most* d *levels we define the advantage*

$$\mathsf{Adv}(\mathcal{A}, \texttt{Gks}^0, \texttt{Gks}^1(\mathcal{S})) := \left|\Pr\left[1 = \mathcal{A} \to \texttt{Gks}^0\right] - \Pr\left[1 = \mathcal{A} \to \texttt{Gks}^1(\mathcal{S})\right]\right|,$$

where $\texttt{Gks}^1(\mathcal{S})$ *is defined in Fig. 11b and* $\texttt{Gks}^0$ *is defined in Fig. 11a.*

4.4 `Xtr` and `Xpd` Key Derivation

The oracles $\mathsf{XPD}_n(h_1, r, args)$ and $\mathsf{XTR}_n(h_1, h_2)$ implement the functionality of the key schedule, i.e., they perform key derivations. Both oracles are parametrized by the name of the key they generate. While XTR (defined in the middle column of Fig. 12) takes the two handles as inputs, the XPD oracles (defined in the left column of Fig. 12) gets a resumption bit and *args* as input in addition to its input handle. The resumption bit is used by the Label function when deriving the bind value to differentiate between application and resumption PSKs.

Throughout this paper, a key k consists of a value and a tag that indicates a hash algorithm, written $\mathsf{alg}(k)$. Recall that keys are tagged and that the xtr and xpd functions use the hash algorithms which is indicated by the tag of their key k, and produce keys with the same tag.

Agile transcript hash (`Hash`) and Diffie-Hellman (`DH`) TLS 1.3 supports a variety of hash algorithm and (elliptic-curve) groups. As for keys and handles, we implement this agility by tagging transcripts and group elements; $\mathsf{alg}(t)$ returns the algorithm to be used for hashing transcript t and $grp(X)$ returns the group for operations on group element X.

To facilitate the proof, we separate the computation of transcript hash values and Diffie-Hellman secrets into their own individual packages. The package $\texttt{Hash}^0$ has an oracle $\mathsf{HASH}(t)$ that evaluates $\mathsf{hash}(t)$ for any supported hash function $\mathsf{alg}(t)$; its straightforward code is given in Appendix A.1. (The proof also involves an idealized package $\texttt{Hash}^1$ that guarantees collision freeness.)

The package `DH`, has two oracles, defined in Fig. 12: DHGEN samples x, stores x, and returns a share $X = g^x$; DHEXP exponentiates a tagged group element Y of the same group with the private exponent of X. The main purpose of this package is to hide private exponents. This will be employed in the proof to justify an idealization step based on an SODH assumption for each supported group. The last line of the DHEXP oracle calls the oracle SET of a key package to record the derived DH secret. Note that a Diffie-Hellman share is honest exactly if it has a key in the exponent table E. DHEXP therefore consults the table also for the share Y to determine its honesty.

$\mathtt{L}_n^P$
$\mathsf{UNQ}_n(h, hon, k)$

if $(\exists\, h^\star : \mathrm{Log}_n[h^\star] = (h', hon', k)$
$\quad \wedge\, \mathsf{level}(h) = r \wedge \mathsf{level}(h^\star) = r')$:
$\quad P(r, hon, r', hon')$
$Log_n[h] \leftarrow (h, hon, k)$
return h

P	the command $P(r, hon, r', hon')$ is
Z	$\emptyset$
A	**if** $hon = hon' = 0 \wedge r = r' = 0$: **throw** *abort*
F	**throw** *abort*

Fig. 13: `L` package

4.5 Keys and Logs

We store keys in `K` packages which contain an array of keys and honesty information, indexed by the handle. See Fig. 14 for the code of `K` and Fig. 13 for the code of `L`. Contrary to other key handles that have a name and a level, the

handles for DH secrets and 0-Keys are used globally across levels. We model this by a separate Nk package (Fig. 15) that has a single array for all keys and a SET_n oracle (without level). To allow for uniform code of $\mathtt{Xtr}_{hs,\ell}$, the $\mathsf{GET}_{dh,\ell}$ oracles of Nk are parametrized by the level but ignore it when retrieving the key.

As discussed in Section 4.3, we use the $\mathtt{K}^b$ package to express security properties. The real game ($b = 0$) stores the concrete key values in an array, whereas the ideal game ($b = 1$) replaces the key values of honest keys with a random value of the same length. In addition, the K package employs the UNQ oracle of package L to model key uniqueness. To this end, L keeps track of all key values passed to UNQ, indexed by their handle and their honesty, and implements different uniqueness guarantees, depending on its pattern parameter P: (1) the F (full) pattern guarantees full uniqueness, causing UNQ to throw an abort if two key values collide, regardless of handle level or honesty; (2) the A (application PSK) pattern guarantees application PSK uniqueness, causing UNQ to throw an abort if two dishonest application PSK values collide, where application PSKs are detected based on the level of their handle; (3) the (Z) (zero) pattern does not provide any guarantees. The proof relies on additional patterns (see Appendix A.2).

The SET oracle guarantees that multiple calls with repeated handles return immediately and do not affect the state of key packages. This allows for multiple calls to SET and together with the close mirroring of key computations in our handle structure guarantees that honest parties with the same handle will also retrieve the same key, irrespective of the order of their SET calls. Finally, note that while the K packages store untagged keys, the GET oracle copies additional information (algorithm and name) from the handle to the key to pass it to the caller as part of the tagged key.

$\mathtt{K}^b_{n,\ell}$
State

K : key table

$\mathsf{SET}_{n,\ell}(h, hon, k^\star)$

assert $\mathsf{name}(h) = n$
assert $\mathsf{level}(h) = \ell$
assert $\mathsf{alg}(k^\star) = \mathsf{alg}(h)$
if $K_{n,\ell}[h] \neq \bot$
  return h
$k \leftarrow \mathsf{untag}(k^\star)$
assert $\mathsf{len}(h) = |k|$
if b :
  if hon :
    $k \leftarrow\!\!\$\ \{0,1\}^{\mathsf{len}(h)}$
$\mathsf{UNQ}_n(h, hon, k)$
$K_{n,\ell}[h] \leftarrow (k, hon)$
return h

$\mathsf{GET}_{n,\ell}(h)$

assert $K_{n,\ell}[h] \neq \bot$
$(k^*, hon) \leftarrow K_{n,\ell}[h]$
$k \leftarrow \mathsf{tag}_h(k^*)$
return (k, hon)

Fig. 14: K package

$\mathtt{Nk}_n$

$\mathsf{SET}_n(h, hon, k)$

assert $\mathsf{name}(h) = n$
if $K_n[h] \neq \bot$
  return h
$\mathsf{UNQ}_n(h, hon, k)$
$K_n[h] \leftarrow (k, hon)$
return h

$\mathsf{GET}_{n,0..d}(h)$

assert $K_n[h] \neq \bot$
$(k, hon) \leftarrow K_n[h]$
return (k, hon)

Fig. 15: Nk package

4.6 Application Key Registration & Honesty

Key *honesty* is a crucial concept to model the guarantee that honest keys, when returned to the adversary, look pseudorandom. A key handle h (which we recall contains the entire derivation history) is marked as honest if either the last PSK

was honest or the last Diffie-Hellman secret was honest. A Diffie-Hellman secret is honest if both shares are honest, i.e., were both generated by DHGEN.

The honesty of an application PSK depends on the adversary's choice when invoking the SET oracle (Fig. 14): If the key is registered as honest ($hon = 1$), a fresh key is sampled instead of the one supplied by the adversary while otherwise the key is used as-is. The handle provided by the adversary is validated for correct structure by the checks **assert** $\mathsf{level}(h) = \ell$ and **assert** $\mathsf{name}(h) = n$ for $\ell = 0$ and $n = psk$.

Also note that the `L` for an application PSK, ensures that the same application PSK is not registered twice, removing modeling redundancy. Recall that, fortunately, the supported hash-algorithms in TLS 1.3 have pairwise-distinct tag lengths, and thus, one cannot register the same PSK value with different algorithms.

`Check`
$\mathsf{XPD}_{n,\ell}(h_1, r, args)$

if $n = bind$:
  if $r = 0$, **assert** $\mathsf{level}(h_1) = 0$
  if $r = 1$, **assert** $\mathsf{level}(h_1) > 0$
elseif $n \in S \cap early$:
  $binder \leftarrow \mathsf{BinderArgs}(args)$
  $h_{binder} \leftarrow \mathsf{BinderHand}(h_1, args)$
  $(k, \_) \leftarrow \mathsf{GET}_{binder,\ell}(h_{binder})$
  assert $binder = k$
elseif $n \in S$:
  $X, Y \leftarrow \mathsf{DhArgs}(args)$
  $h_{dh} \leftarrow \mathsf{DhHand}(h_1)$
  assert $h_{dh} = \mathsf{dh}\langle\mathsf{sort}(X, Y)\rangle$
  $binder \leftarrow \mathsf{BinderArgs}(args)$
  $h_{binder} \leftarrow \mathsf{BinderHand}(h_1, args)$
  $(k, \_) \leftarrow \mathsf{GET}_{binder,\ell}(h_{binder})$
  assert $binder = k$
$h \leftarrow \mathsf{XPD}_{n,\ell}(h_1, r, args)$
return h

Fig. 16: Check code

4.7 Front-End Check

The XPD oracle of the `Check` package, see Fig. 16 filters calls to `Xpd` and enforces usage restrictions for the key schedule that the TLS 1.3 handshake guarantees and that key schedule security relies on. `Check` ensures that the resumption flag is consistent with the level of the PSK; that the shares contained in the transcript correspond to the shares used in the DH computation, and that the binder tag included in the transcript of later stages (at the end of the last ClientHello message) is the same that was computed and checked in the early stage. We need `Check` to enforce this property at (at least) one node—called *separation point*—on the path to each output key.

Definition 4.4 (Separation Points).
For a key schedule $ks = (N, \mathsf{Label}, \mathsf{PrntN})$*, we call* $S \subseteq N$ *a* set of separation points*, if it satisfies the following two requirements:*

- *For each* $n \in O$ *the path from psk to* n *contains an* $n' \in S$*.*
- *If there exists a path from dh to a* $n \in O$*, then it contains an* $n' \in S$*.*

In addition, to express our assumptions, for each xpd operation in the key schedule, we choose one representative child. I.e., $XPR \subseteq N$ is a *representative set* for ks if $psk, esalt \in XPR$ and for each name $n \in N$ with only a single parent (these are the xpd nodes), either n or exactly one sibling of n is contained in XPR.

4.8 Key Schedule Theorem

Theorem 4.5. *Let ks be a TLS-like key schedule with representative set XPR and a set of separation points S. Let* $d \in \mathbb{N}$*. Let* $PO^* := \{n : \exists n' \in O^* : (n, \_) =$

$\mathsf{PrntN}(n')\}$ *and* $SO^* := \bigcup\{\mathsf{ChldrnN}(n_1) \mid n_1 \in PO^*\}$. *There exists an efficient simulator* $\mathcal{S}$ *such that for all adversaries* $\mathcal{A}$ *which make queries for at most* d *resumption levels, use at most* $s_{n,\ell,alg} = t^{hon=1}_{n,alg}$ *honest and* $t^{hon=0}_{n,alg}$ *dishonest parent keys for algorithm* alg *to generate keys with name* n *at level* ℓ, *and let* $t_{n,alg}$ *be* $\max\{t^{hon=0}_{n,alg}, t^{hon=1}_{n,alg}\}$.

$$
\begin{aligned}
&\mathsf{Adv}(\mathcal{A}, \mathtt{Gks}^0, \mathtt{Gks}^1(\mathcal{S})) \leq \\
&\sum_{alg\in\mathcal{H}} \Big(\mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{main}_{cr} \to \mathcal{R}_{alg,\mathsf{hash}}, \mathtt{Gcr}^{\mathsf{hash}\text{-}alg,b}) \\
&+ \sum_{j\in\{Z,D\},\mathsf{f}\in\{\mathsf{xtr},\mathsf{xpd}\}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{main}_{j} \to \mathcal{R}_{alg,\mathsf{f}}, \mathtt{Gcr}^{\mathsf{hash}\text{-}alg,b})\Big) \\
&+ \max_{i\in\{0,1\}} \Big[\sum_{grp\in\mathcal{G}} \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{sodh} \to \mathcal{R}^{grp}_{sodh2}, \mathtt{Gsodh2}^{grp,b}) \\
&+ 2 \cdot \min_{alg\in\mathcal{H}} \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{sodh} \to \mathcal{R}^{alg}_{sodh\text{-}cr}, \mathtt{Gcr}^{\mathsf{hash}\text{-}alg,b}) \\
&+ \sum_{alg\in\mathcal{H}} \Big(\sum_{0\leq\ell\leq d} \big[s_{es,\ell,alg} \cdot \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{es,\ell,alg}, \mathtt{Gpr}^{\mathsf{xtr}\text{-}alg^\dagger,b}) \\
&+ s_{hs,\ell,alg} \cdot \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{hs,\ell,alg}, \mathtt{Gpr}^{\mathsf{xtr}\text{-}alg,b}) \\
&+ s_{as,\ell,alg} \cdot \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{as,\ell,alg}, \mathtt{Gpr}^{\mathsf{xtr}\text{-}alg,b}) \\
&+ \sum_{n\in XPR} s_{n,\ell,alg} \cdot \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{n,\ell,alg}, \mathtt{Gpr}^{\mathsf{xpd}\text{-}alg,b})\big] \\
&+ 2 \cdot \Big[\Big(\sum_{0\leq\ell\leq d} s_{esalt,\ell,alg}\Big) \cdot \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{esalt,pi,alg}, \mathtt{Gpr}^{\mathsf{xpd}\text{-}alg,b}) \\
&\qquad + \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{esalt,pi\text{-}cr,alg}, \mathtt{Gcr}^{\mathsf{hash}\text{-}alg,b})\Big] \\
&+ \sum_{n\in SO^*\cap XPR} 2 \cdot \Big[\Big(\sum_{0\leq\ell\leq d} s_{n,\ell,alg}\Big) \cdot \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{n,pi,alg}, \mathtt{Gpr}^{\mathsf{xpd}\text{-}alg,b}) \\
&\qquad + \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{n,pi\text{-}cr,alg}, \mathtt{Gcr}^{\mathsf{hash}\text{-}alg,b})\Big] \\
&+ 2^{-\mathsf{len}(alg)} \cdot \Big[(c+1)\cdot t^2_{esalt,alg} + (2c+6)\cdot t^2_{es,alg} + \sum_{n\in PO*} ((2c+6)\cdot t^2_{n,alg})\Big]\Big)\Big],
\end{aligned}
$$

$\mathcal{A}_i$ *defined in Appendix D.1, modifies* $\mathcal{A}$ *without significantly changing its complexity: it behaves as* $\mathcal{A}$ *except that it returns* i *on some aborts;* $\mathcal{R}^{main}_{*} := \mathcal{R}^{ch\text{-}map} \to \mathcal{R}_{*}$ *when replacing* $*$ *by cr, Z, D or sodh;* $\mathcal{R}^{main}_{*,alg} := \mathcal{R}^{ch\text{-}map} \to \mathcal{R}^{alg}_{*}$ *when replacing* $*$ *by* n, pi, $n, pi\text{-}cr$, $esalt, pi$ *or* $esalt, pi\text{-}cr$; $\mathcal{R}^{main}_{*,\ell,alg} := \mathcal{R}^{ch\text{-}map} \to \mathcal{R}_{*,\ell} \to \mathcal{R}^{alg}_{*,\ell}$ *when replacing* $*$ *by es, hs, as, n; the simulator* $\mathcal{S}$ *is marked in grey in Fig. 26b;* $\mathcal{R}_{alg,\mathsf{f}}$ *is defined in Lemma A.2,* $\mathcal{R}_{sodh}$ *is defined in Fig. 32a,* $\mathcal{R}_{es,\ell}$ *is defined in Fig. 34a,* $\mathcal{R}_{hs,\ell}$ *and* $\mathcal{R}_{as,\ell}$ *are defined analogously, and* $\mathcal{R}_{n,\ell}$ *for* $n \in XPR$ *and* $0 \leq \ell \leq d$ *are defined in Fig. 34b;* $\mathcal{R}^{alg}_{n,\ell}$ *for* $n \in XPR \cup \{es, hs, as\}$ *are defined in Lemma E.6;* $\mathcal{R}^{alg}_{sodh\text{-}cr}$ *and* $\mathcal{R}^{grp}_{sodh2}$ *are defined in Lemma E.1,* $\mathcal{R}_{esalt,pi}$ *is defined in Fig. 32c and* $\mathcal{R}_{n\in PO^*,pi}$ *is defined in Fig. 32d;* c *is a small constant which depends on the min-entropy of the distribution* $\mathsf{xtr}(k, U_{\mathsf{len}(alg)})$, *where* k *is a fixed key and* $U_{2^{\mathsf{len}(alg)}}$ *denotes the uniform distribution of strings of length* $\mathsf{len}(alg)$.

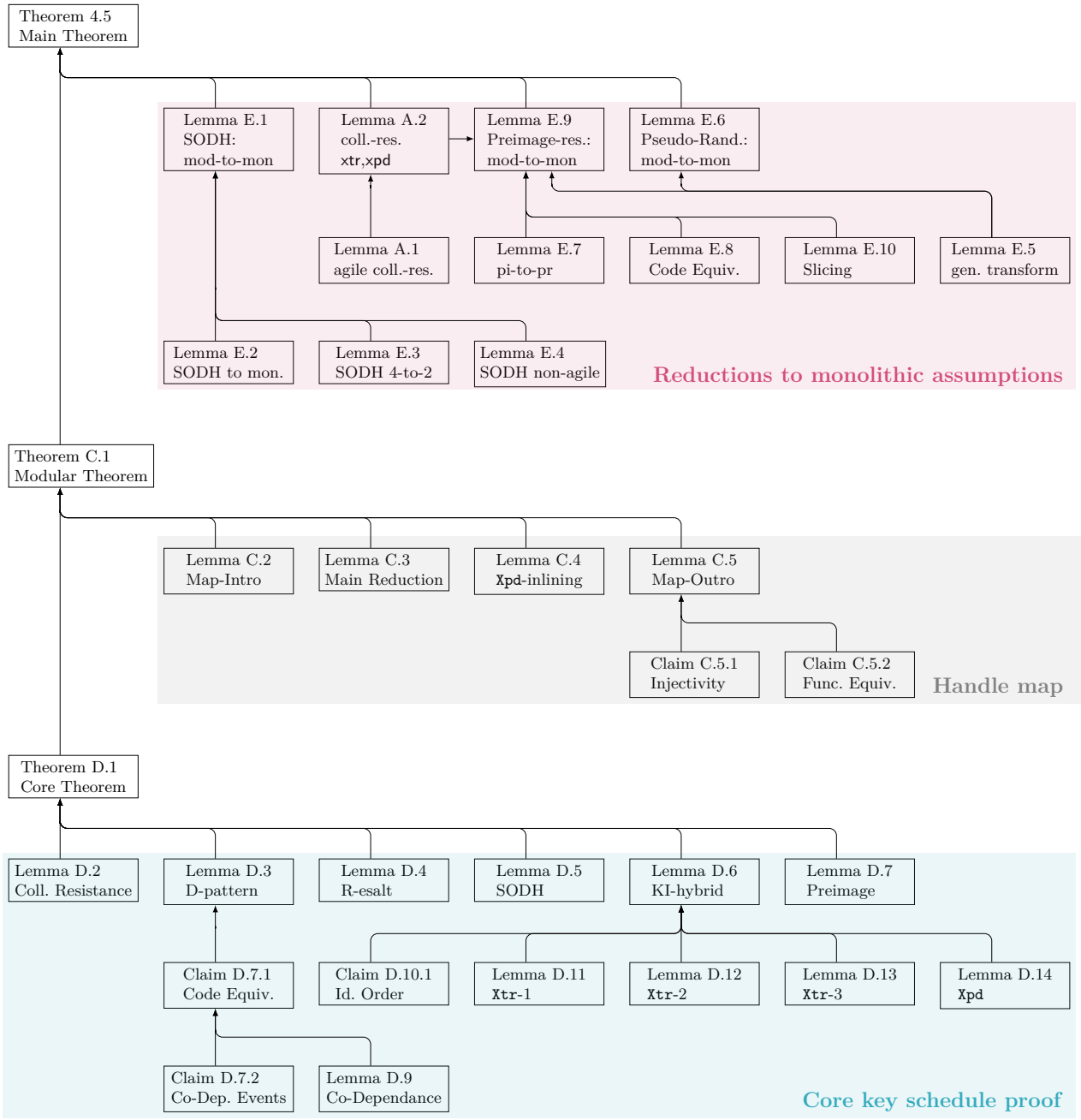


Fig. 17: Proof Tree

5 From Key Schedule to Key Exchange and TLS

To complement our analysis of the key schedule, we return to its integration within TLS 1.3. We re-state the handshake protocol in terms of calls to the concrete key schedule interface, and give an informal argument of key-exchange security based on the idealized key schedule, following the general proof structure of the SIGMA-I paradigm. Fig. 18 describes the TLS 1.3 handshake, refining the outline at the top of Fig. 1, but still omitting details. Its code delegates all key derivations to the key schedule to obtain keys at each step of the protocol. It uses some of these keys, and returns the others to the record layer: nextKeys advances the key-schedule to derive traffic secrets for 0-RTT data (k_{cet}), for handshake messages (k_{cht}, k_{sht}), and for 1-RTT data (k_{cat}, k_{sat}); and two exporter secrets (k_{eem}, k_{eam}).

Initially, the client offers a *list* of shares $\{g_j^{x_j}\}$ over disjoint groups g_j (identified here by their tagged generators) each produced by gendh (Section 4.4). The client may also indicate support for groups without offering shares, but this can be ignored from the key schedule perspective, as it just involves an extra roundtrip (called hello-retry-request) to get the share. The client also offers a list of pre-shared key objects $k_{psk,i}$, each associated with a label $label_i$ and tagged with its hash algorithm $\mathsf{alg}(k_{psk,i})$ and a flag $\mathsf{res}(k_{psk,i})$ that indicates whether it refers to an application PSK or an internal resumption PSK. Although the PSK label only represents its local name, the binder k^i_{binder} (computed as the HMAC of the client's offer under a key derived from $k_{psk,i}$ with either a `ext binder` or `res binder` label) guarantees that the peers agree on the flag, the label and the key value when a PSK is selected. The distinction between application and resumption PSKs matters because they imply different identities. For resumption PSK, it is the certificate in the original pure Diffie-Hellman handshake from which the PSK originates. Without the `ext` or `res` label, a malicious server A can install the resumption PSK of a session as an application PSK at another (honest) server B, allowing A to forward a resumption from C to B (with mixed `psk_dhe_ke` key exchange, and thus honest keys) without agreement on the peer's identity. Our proof addresses this issue by mapping PSKs, at a significant complexity cost. The list of PSK can be empty, leaving the hash algorithm undefined. Our model handles this without loss of generality, by adding a dishonest PSK shared by all clients and servers for each algorithm. This matches the concrete key computation, which uses a string of 0s of the hash algorithm's digest length if no PSK is provided. The list of Diffie-Hellman shares can also be empty, causing TLS to use a default Diffie-Hellman secret built as a bitstring of zeros (whose length matches the hash algorithm digests). This is similarly modelled by adding unsafe singleton group descriptors. Upon receiving the client hello (CH) message, the server chooses one of the offered PSKs and one of the offered shares, checks the binder for the selected PSK, and indicates its choice in the server hello message (SH). 0-RTT is permitted only if the server picks the first offered psk_0. If the server picks any other PSK, the client discards its early secret and restarts with a fresh key schedule instance. If the server sends a certificate (C), it signs the transcript containing the hellos

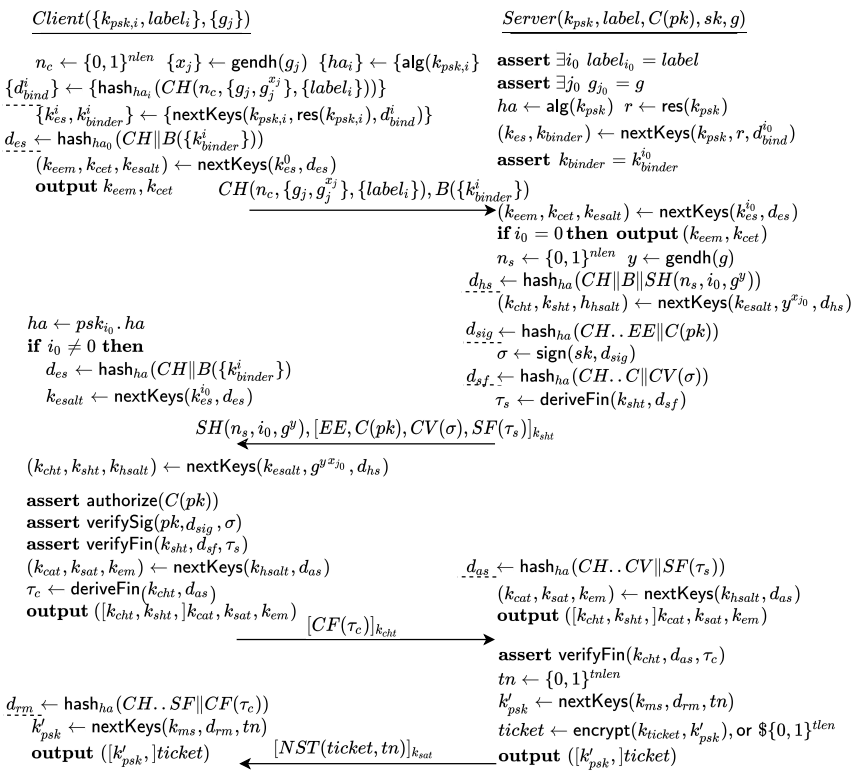


Fig. 18: TLS 1.3 handshake, built on top of the key schedule. Functions gendh and nextKeys model key-schedule computations. Following the standard, handshake traffic secrets are used to derive both finished messages (deriveFin) and the record keys for protecting handshake messages $[\dots]_{k_{cht}}$. We mark output keys that must not be revealed to a key-exchange adversary by $[\dots]$. QUIC uses handshake traffic secrets, and thus cannot be composed with TLS 1.3 generically. Application traffic secrets are used to encrypt NST messages, and thus cannot be used in generic compositions. Hence, only the exporter secrets enjoys Bellare-Rogaway key-indistinguishability [9].

(CH, SH), binders, encrypted extensions (EE) and certificate (C), and sends the signature in the certificate verify (CV) message. It finally creates and sends a MAC in the server finished (SF) message. Interestingly, the application traffic key is derived from a transcript that only contains the server's finished MAC, before the client confirms the server (and its own) identity, a pattern sometimes called 0.5-RTT. Once the client finished MAC is created, the handshake outputs a fresh PSK to resume the connection in the future, using the full transcript up to the client finished (CF) message to derive the resumption secret and a ticket nonce to derive individual resumption PSKs. Interestingly, some TLS messages (including the `NewSessionTicket` that carry resumption tickets) are sent after the completion of the handshake.

Towards a Handshake Security Model In key-exchange, two sessions are *partnered* if they have the same key handles, and a session is *fresh* if all its keys are honest and neither the session itself not its partner session were revealed. With this definition, key-indistinguishability of the key-exchange follows conceptually from indistinguishability of the key schedule. In practice, this notion of freshness is too limited. With partnering, it implicitly authenticates transcript and PSK identities, but does not capture certificate-based authentication (which uses long-term secrets), and does not provide explicit key confirmation (without interpreting the certificate verify and finished messages in the transcript). I.e., one needs to extend the model to capture the derivation of finished keys from the handshake traffic secrets (with a standard KDF), and the computation of finished MAC and transcript signatures (with standard UNF-CMA). The uniqueness property of handles then suffices to prove agreement across resumption levels for the TLS 1.3 handshake, regardless of the handshake mode (PSK or mixed) in the session history. Our key-schedule model allow for adversarially chosen application PSKs and ephemeral Diffie-Hellman shares, thus enables adaptive corruption of signing keys in the key exchange. If one considers signing keys as long-term keys and PSKs as ephemeral, then our model is stronger than perfect forward secrecy, since (a) we have security *before* the long-term signing key is corrupted, (b) we have security against passive adversaries, but also (c) we allow for adversarially chosen ephemeral values and (d) we have security if the adversary did not tamper with the peer's share.

6 Lessons Learned & Afterthoughts on the Key Schedule

We now discuss changes to the key schedule that would improve its security and simplify its analysis and may be of independent interest for other protocols.

Simplify SODH. The salted Diffie-Hellman computation extracts entropy from the DH secret and mixes it with the PSK-derived salt (which is under adversarial influence). A separate DH extraction, preferably hashing the (sorted) public shares together with the secret, followed by a dual PRF, would enable a proof based on the simpler and better understood Oracle Diffie-Hellman assumption. The hashing of shares would also remove the need to map DH secrets (currently computable from multiple pairs of shares), and would enable the use of a more abstract functionality such as a CCA-secure KEM (as in TLS 1.2 [17]). These changes would thus also ease the integration of post-quantum secure primitives.

Eliminate PSK mapping. Similarly, *directly* applying domain-separation for computations based on application and resumption PSKs via distinct labels would remove the need to map PSKs and argue via inclusion of binders at separation points indirectly. Both proposals follow the same design pattern: first sanitize input key materials, to prevent malleability (DH secrets) and collisions (dishonest resumption PSKs and adversarially-chosen application PSKs).

Avoid Agile Assumptions Our development supports multiple hash algorithms without requiring any hash-agile assumptions, by observing that the hash functions currently used by TLS 1.3 have pairwise-distinct digest lengths. This

is brittle, e.g. adding support for SHA3 with the same lengths as SHA2 would require to formally account for cross-algorithm collisions. This may be prevented by tagging the outputs of all extractors and KDFs with hash algorithms. Similarly, we may avoid the current need for agile (S)ODH assumptions by tagging group elements with both a group descriptor and a single extraction algorithm.

Prevent PSK Reflections. Drucker and Gueron note that TLS 1.3 is subject to reflection attacks due to its symmetric use of PSKs [38]. Hence, in our model, the same PSK handle may either be used by two parties, as intended, or by the same party acting both as a client and as a server. This is a security risk, inasmuch as applications may embed identity information in PSK identifiers to benefit from their early authentication. It may also enable key synchronization attacks and other variants of key compromise impersonation [16] when identities are also symmetrical. When using PSKs, the standard unfortunately forbids certificate-based authentication, which would otherwise provide more detailed, role-specific identity information. At the key schedule level, it may be possible to enforce better separation by tagging PSK identifiers with roles.

Enforce Stronger Modularity. Applied cryptographers often complain that, in TLS 1.2, the subtle interleaving of the handshake with the record layer hinders its analysis based on the well-established Bellare-Rogaway [9] security model [45]. While TLS 1.3 tries to enforce cleaner separation between handshake and record keys, it still fails in some important places. Notably, the handshake traffic secrets, meant to be released to the record layer (be it TLS, DTLS, or QUIC) are also used by the handshake to derive finished keys. Similarly, some handshake messages are encrypted under keys derived from application traffic secrets (e.g. New Session Ticket, carrying resumption PSKs, late client authentication, and key updates). This complicates the modeling of data stream security, as application data may be interleaved with handshake messages (e.g. the same QUIC packet may contain both data and session tickets). To prevent such issues, and many others, we suggest the RFC document more explicitly its application interface.

Acknowledgements

We thank Benjamin Dowling for collaboration at the early stages of this work during his internship at Microsoft Research Cambridge and insightful follow-up discussions relating our work to his own line of research.

References

1. C. Abate, P. G. Haselwarter, E. Rivas, A. V. Muylder, T. Winterhalter, C. Hriţcu, K. Maillard, and B. Spitters. Ssprove: A foundational framework for modular cryptographic proofs in coq. Cryptology ePrint Archive, Report 20201/397, 2021. `https://eprint.iacr.org/2021/397`.
2. M. Abdalla, M. Bellare, and P. Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In D. Naccache, editor, *CT-RSA 2001*, volume 2020 of *LNCS*, pages 143–158. Springer, Heidelberg, Apr. 2001.

3. D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, and P. Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In I. Ray, N. Li, and C. Kruegel, editors, *ACM CCS 2015*, pages 5–17. ACM Press, Oct. 2015.
4. N. J. AlFardan and K. G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *2013 S&P*, pages 526–540. IEEE, May 2013.
5. G. Arfaoui, X. Bultel, P.-A. Fouque, A. Nedelcu, and C. Onete. The privacy of the TLS 1.3 protocol. *PoPETs*, 2019(4):190–210, Oct. 2019.
6. N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J. A. Halderman, V. Dukhovni, E. Käsper, S. Cohney, S. Engels, C. Paar, and Y. Shavitt. DROWN: Breaking TLS using SSLv2. In T. Holz and S. Savage, editors, *USENIX Security 2016*, pages 689–706. USENIX, Aug. 2016.
7. C. Badertscher, C. Matt, U. Maurer, P. Rogaway, and B. Tackmann. Augmented secure channels and the goal of the TLS 1.3 record layer. In M. H. Au and A. Miyaji, editors, *ProvSec 2015*, volume 9451 of *LNCS*, pages 85–104. Springer, Heidelberg, Nov. 2015.
8. M. Bellare. New proofs for NMAC and HMAC: Security without collision resistance. *Journal of Cryptology*, 28(4):844–878, Oct. 2015.
9. M. Bellare and P. Rogaway. Entity authentication and key distribution. In D. R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249. Springer, Heidelberg, Aug. 1994.
10. M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, May / June 2006.
11. B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue. A messy state of the union: Taming the composite state machines of TLS. In *2015 S&P*, pages 535–552. IEEE, May 2015.
12. K. Bhargavan, B. Blanchet, and N. Kobeissi. Verified models and reference implementations for the TLS 1.3 standard candidate. In *2017 S&P*, pages 483–502. IEEE, May 2017.
13. K. Bhargavan, C. Brzuska, C. Fournet, M. Green, M. Kohlweiss, and S. Zanella-Béguelin. Downgrade resilience in key-exchange protocols. In *2016 S&P*, pages 506–525. IEEE, May 2016.
14. K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, J. Pan, J. Protzenko, A. Rastogi, N. Swamy, S. Zanella-Béguelin, and J.-K. Zinzindohoué. Implementing and proving the TLS 1.3 record layer. In *IEEE Security & Privacy*. IEEE, 2017.
15. K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Pironti, and P.-Y. Strub. Triple handshakes and cookie cutters: Breaking and fixing authentication over tls. In *IEEE Symposium on Security & Privacy (Oakland)*, 2014.
16. K. Bhargavan, A. Delignat-Lavaud, and A. Pironti. Verified contributive channel bindings for compound authentication. In *NDSS 2015*. ISOC, Feb. 2015.
17. K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and S. Zanella Béguelin. Proving the TLS handshake secure (as it is). In J. A. Garay and R. Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 235–255. Springer, Heidelberg, Aug. 2014.
18. K. Bhargavan and G. Leurent. Transcript collision attacks: Breaking authentication in TLS, IKE and SSH. In *NDSS 2016*. ISOC, Feb. 2016.
19. B. Blanchet. CryptoVerif: Computationally sound mechanized prover for cryptographic protocols. In *Formal Protocol Verification*, volume 117, page 156, 2007.

20. B. Blanchet. Composition theorems for CryptoVerif and application to TLS 1.3. In *CSF*, pages 16–30, July 2018.
21. B. Blanchet, B. Smyth, V. Cheval, and M. Sylvestre. ProVerif 2.00: automatic cryptographic protocol verifier. User Manual, 2018.
22. H. Böck, J. Somorovsky, and C. Young. Return of bleichenbacher's oracle threat (ROBOT). In W. Enck and A. P. Felt, editors, *USENIX Security 2018*, pages 817–849. USENIX, Aug. 2018.
23. D. Boneh. The decision Diffie-Hellman problem. In *ANTS*, volume 1423 of *LNCS*. Springer, Heidelberg, 1998.
24. J. Brendel, M. Fischlin, F. Günther, and C. Janson. PRF-ODH: Relations, instantiations, and impossibility results. In J. Katz and H. Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 651–681. Springer, Heidelberg, Aug. 2017.
25. R. Bricout, S. Murphy, K. G. Paterson, and T. van der Merwe. Analysing and exploiting the mantin biases in RC4. Cryptology ePrint Archive, Report 2016/063, 2016. `http://eprint.iacr.org/2016/063`.
26. C. Brzuska, E. Cornelissen, and K. Kohbrok. Cryptographic security of the MLS rfc, draft 11. Cryptology ePrint Archive, Report 20201/137, 2021. `https://eprint.iacr.org/2021/137`.
27. C. Brzuska, A. Delignat-Lavaud, C. Fournet, K. Kohbrok, and M. Kohlweiss. State separation for code-based game-playing proofs. In T. Peyrin and S. Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 222–249. Springer, Heidelberg, Dec. 2018.
28. C. Brzuska, M. Fischlin, N. Smart, B. Warinschi, and S. Williams. Less is more: Relaxed yet composable security notions for key exchange. Cryptology ePrint Archive, Report 2012/242, 2012. `http://eprint.iacr.org/2012/242`.
29. C. Brzuska, K. Kohbrok, and M. Kohlweiss. From the Monolithic Age to Agile Composite Assumptions. Preprint, Feb. 2021.
30. S. Chen, S. Jero, M. Jagielski, A. Boldyreva, and C. Nita-Rotaru. Secure communication channel establishment: TLS 1.3 (over TCP fast open) vs. QUIC. In K. Sako, S. Schneider, and P. Y. A. Ryan, editors, *ESORICS 2019, Part I*, volume 11735 of *LNCS*, pages 404–426. Springer, Heidelberg, Sept. 2019.
31. C. Cremers, M. Horvat, J. Hoyland, S. Scott, and T. van der Merwe. A comprehensive symbolic analysis of TLS 1.3. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *ACM CCS 2017*, pages 1773–1788. ACM Press, 2017.
32. C. Cremers, M. Horvat, S. Scott, and T. van der Merwe. Automated analysis and verification of TLS 1.3: 0-RTT, resumption and delayed authentication. In *2016 S&P*, pages 470–485. IEEE, May 2016.
33. H. Davis and F. Günther. Tighter proofs for the sigma and tls 1.3 key exchange protocols. Cryptology ePrint Archive, Report 2020/1029, 2020. `https://eprint.iacr.org/2020/1029`.
34. D. Diemert and T. Jager. On the tight security of tls 1.3: Theoretically-sound cryptographic parameters for real-world deployments. Cryptology ePrint Archive, Report 2020/726, 2020. `https://eprint.iacr.org/2020/726`.
35. B. Dowling, M. Fischlin, F. Günther, and D. Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In I. Ray, N. Li, and C. Kruegel, editors, *ACM CCS 2015*, pages 1197–1210. ACM Press, Oct. 2015.
36. B. Dowling, M. Fischlin, F. Günther, and D. Stebila. A cryptographic analysis of the TLS 1.3 draft-10 full and pre-shared key handshake protocol. Cryptology ePrint Archive, Report 2016/081, 2016. `http://eprint.iacr.org/2016/081`.

37. B. Dowling and D. Stebila. Modelling ciphersuite and version negotiation in the TLS protocol. In E. Foo and D. Stebila, editors, *ACISP 15*, volume 9144 of *LNCS*, pages 270–288. Springer, Heidelberg, June / July 2015.
38. N. Drucker and S. Gueron. Selfie: reflections on TLS 1.3 with PSK. Cryptology ePrint Archive, Report 2019/347, 2019. `https://eprint.iacr.org/2019/347`.
39. F. Dupressoir, K. Kohbrok, and S. Oechsner. Bringing state-separating proofs to easycrypt - a security proof for cryptobox. Cryptology ePrint Archive, Report 20201/326, 2021. `https://eprint.iacr.org/2021/326`.
40. M. Fischlin and F. Günther. Replay attacks on zero round-trip time: The case of the TLS 1.3 handshake candidates. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 60–75. IEEE, 2017.
41. M. Fischlin, F. Günther, G. A. Marson, and K. G. Paterson. Data is a stream: Security of stream-based channels. In R. Gennaro and M. J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 545–564. Springer, Heidelberg, Aug. 2015.
42. M. Fischlin, F. Günther, B. Schmidt, and B. Warinschi. Key confirmation in key exchange: A formal treatment and implications for TLS 1.3. In *2016 S&P*, pages 452–469. IEEE, May 2016.
43. P. Gaži, K. Pietrzak, and M. Rybár. The exact PRF-security of NMAC and HMAC. In J. A. Garay and R. Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 113–130. Springer, Heidelberg, Aug. 2014.
44. J. Iyengar and M. Thomson. QUIC. IETF draft, 2019.
45. T. Jager, F. Kohlar, S. Schäge, and J. Schwenk. Authenticated confidential channel establishment and the security of TLS-DHE. *Journal of Cryptology*, 30(4):1276–1324, Oct. 2017.
46. M. Kohlweiss, U. Maurer, C. Onete, B. Tackmann, and D. Venturi. (De-)constructing TLS 1.3. In A. Biryukov and V. Goyal, editors, *INDOCRYPT 2015*, volume 9462 of *LNCS*, pages 85–102. Springer, Heidelberg, Dec. 2015.
47. H. Krawczyk. SIGMA: The "SIGn-and-MAc" approach to authenticated Diffie-Hellman and its use in the IKE protocols. In D. Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 400–425. Springer, Heidelberg, Aug. 2003.
48. H. Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 631–648. Springer, Heidelberg, Aug. 2010.
49. H. Krawczyk. A unilateral-to-mutual authentication compiler for key exchange (with applications to client authentication in TLS 1.3). In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *ACM CCS 2016*, pages 1438–1450. ACM Press, Oct. 2016.
50. H. Krawczyk and H. Wee. The OPTLS protocol and TLS 1.3. Cryptology ePrint Archive, Report 2015/978, 2015. `http://eprint.iacr.org/2015/978`.
51. X. Li, J. Xu, Z. Zhang, D. Feng, and H. Hu. Multiple handshakes security of TLS 1.3 candidates. In *2016 S&P*, pages 486–505. IEEE, May 2016.
52. G. Lowe. A hierarchy of authentication specification. In *10th Computer Security Foundations Workshop (CSFW '97)*, pages 31–44. IEEE Computer Society, 1997.
53. U. Maurer. Constructive cryptography - a primer (invited paper). In R. Sion, editor, *FC 2010*, volume 6052 of *LNCS*, page 1. Springer, Heidelberg, Jan. 2010.
54. N. Mavrogiannopoulos, F. Vercauteren, V. Velichkov, and B. Preneel. A cross-protocol attack on the TLS protocol. In T. Yu, G. Danezis, and V. D. Gligor, editors, *ACM CCS 2012*, pages 62–72. ACM Press, Oct. 2012.
55. K. G. Paterson and T. van der Merwe. Reactive and proactive standardisation of TLS. In *Security Standardisation Research*, pages 160–186, 2016.

56. C. Patton and T. Shrimpton. Partially specified channels: The TLS 1.3 record layer without elision. Cryptology ePrint Archive, Report 2018/634, 2018.
57. P. Rogaway and Y. Zhang. Simplifying game-based definitions - indistinguishability up to correctness and its application to stateful AE. In H. Shacham and A. Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 3–32. Springer, Heidelberg, Aug. 2018.

$\mathsf{hash}(t)$

$t^* \leftarrow \mathsf{untag}(t)$
$alg \leftarrow \mathsf{alg}(t)$
$d^* \leftarrow \mathsf{hash}\text{-}alg(t^*)$
$d \leftarrow \mathsf{tag}_{alg}(d^*)$
return d

$\mathsf{hmac}_{alg}(k, t)$

$ipad \leftarrow 0x36^{\mathsf{blocksize}(alg)}$
$opad \leftarrow 0x5C^{\mathsf{blocksize}(alg)}$
$k^\dagger \leftarrow k \| 0^{|ipad|-|k|}$
$inner \leftarrow \mathsf{hash}\text{-}alg(k^\dagger \oplus ipad) \| t$
$d \leftarrow \mathsf{hash}\text{-}alg((k^\dagger \oplus opad) \| inner)$
return d

$\mathsf{xtr}(k_1, k_2)$

if $\mathsf{alg}(k_1) = \perp$:
  $alg \leftarrow \mathsf{alg}(k_2)$
  $k_2 \leftarrow \mathsf{untag}(k_2)$
else $alg \leftarrow \mathsf{alg}(k_1)$
  $k_1 \leftarrow \mathsf{untag}(k_1)$
$k \leftarrow \mathsf{hmac}_{alg}(k_1, k_2)$
return $\mathsf{tag}_{alg}(k)$

$\mathsf{xtr}\text{-}alg(k_1, k_2)$

$k \leftarrow \mathsf{hmac}_{alg}(k_1, k_2)$
return k

$\mathsf{xpd}\text{-}alg(k_1, (label, d))$

if $label = []$:
  return $\mathsf{hmac}_{alg}(k_1, d)$
$t \leftarrow (\mathsf{len}(alg), \texttt{tls13} \ \| label, d) \| 0x01$
$k \leftarrow \mathsf{hmac}_{alg}(k_1, t)$
return k

$\mathsf{xpd}(k_1, (label, d))$

$alg \leftarrow \mathsf{alg}(k_1)$
$k_1 \leftarrow \mathsf{untag}(k_1)$
$k \leftarrow \mathsf{xpd}\text{-}alg(k_1, label, d)$
return $\mathsf{tag}_{alg}(k)$

Fig. 19: Specification of the $\mathsf{hash}, \mathsf{hmac}_{alg}, \mathsf{xtr}$ and xpd algorithms according to standard. Recall from Fig. 1 that xpd with empty label executes hmac_{alg}. Tuple notation $(.,.)$ denotes an injective encoding of pairs, while concatenation $.||.$ is a non-injective operation on pairs of strings. Observe that hmac_{alg} pads each input key to blocklength by adding zeroes. We assume that blocklength is always greater than algorithm length. For Sha-256, for example, the blocklength is 512 bits. The blocklength of an algorithm is determined by the cipher it internally iterates.

A Agile and Composed Assumptions and Notions

Recall that TLS 1.3 supports multiple algorithms for xtr [8] (used to extract keys and MAC tags) and xpd [48]. Therefore, our proof for the TLS 1.3 key schedule (see Appendix 4) also relies on assumptions for multiple algorithms, so-called *agile* assumptions. We model agile assumptions by agile functions. Their inputs are tagged with an algorithm, and the agile function executes the algorithm in the tag on the input. For details, see Section 4.4. Note that there, we also discuss agility for supporting multiple groups in Diffie-Hellman computations.

Additionally, our proof of the TLS 1.3 key schedule relies on defining reductions via cuts in graphs as illustrated in Fig. 6 and Fig. 6 in Section 2. Thus, in this appendix, we formalize all assumptions via security games which we write as a *composition* of several packages. Of particular important will be Key and Log packages which conceptually play the same role as the K and L packages introduced in Section 4.5. However, in the proof and assumptions, we rely on a greater variety of Key and Log packages with different abort patterns than just A, and F. We introduce our various Key and Log packages in Appendix A.2.

We discuss agile collision-resistance in Appendix A.1. Section A.3 contains several security games for xtr and xpd, including the SODH assumption. In Appendix E, we then reduce our composed assumptions to the monolithic assumptions that were introduced in Section 3.

A.1 Agile Collision-Resistance

We reduce the collision-resistance of xtr and xpd to the collision-resistance of hash. The following result is folklore and the closest attribution we could find is a talk by Hugo Krawczyk at the 8th BIU Winter School on Cryptography[7]. We refer to Fig. 19 for the specification of xtr and xpd.

Lemma A.1 (Collision Resistance of xtr and xpd). *For all adversaries $\mathcal{A}$, $\mathsf{f} \in \{\mathsf{xtr}, \mathsf{xpd}\}$ and any $alg \in \mathcal{H}$*

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gcr}^{\mathsf{f}\text{-}alg,b}) \leq \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{\mathsf{f}}, \mathtt{Gcr}^{\mathsf{hash}\text{-}alg,b})$$

Fig. 20 defines the collision-resistance game $\mathtt{Gacr}^{\mathsf{f},b}$ for a given agile function f, with a table *Hash* that records input texts t and their digests $d = \mathsf{f}(t)$. In the ideal game ($b = 1$), the oracle aborts when the adversary submits a text whose digest collides with a prior entry in the table. We denote the collision-resistance advantage of an adversary $\mathcal{A}$ against agile function f by $\mathsf{Adv}(\mathcal{A}, \mathtt{Gacr}^{\mathsf{f},b})$. Recall that f can be any out of the set $\{\mathsf{hash}, \mathsf{xtr}, \mathsf{xpd}\}$. When $\mathsf{f} = \mathsf{hash}$, then we introduce the following abbreviation:

$$\mathtt{Hash}^b := \mathtt{Gacr}^{\mathsf{hash},b} \text{ for } b \in \{0, 1\}$$

[7] https://cyber.biu.ac.il/wp-content/uploads/2017/08/KE3_Hugo_BIU_Feb2018.pdf

The difference between $\mathtt{Gacr}^{\mathsf{f},b}$ and the collision-resistance game $\mathtt{Gcr}^{\mathsf{f}\text{-}alg,b}$, given in Fig. 7 in Section 3.1, is that $\mathtt{Gacr}^{\mathsf{f},b}$ is parametrized by a set of supported algorithms $\mathcal{H}$ and that the HASH oracle of $\mathtt{Gacr}^{\mathsf{f},b}$ asserts that the tag of the input is indeed in the set $\mathcal{H}$. Note that $\mathcal{H}$ can be any set of algorithms (as long as the computational assumptions hold for that set). Lemma A.2 relies on $\mathcal{H}$ containing only hash-algorithms with different-length outputs. All other statements in this article (as long as they do not rely on Lemma A.2) do not require any specific properties from $\mathcal{H}$. We now relate that agile collision-resistence to non-agile collision-resistance (see Fig. 7 in Section 3.1).

$\underline{\underline{\mathtt{Gacr}^{\mathsf{f},b}}}$
$\mathsf{HASH}(t)$

assert $\mathsf{alg}(t) \in \mathcal{H}$
if $Hash[t] \neq \bot$:
  return $Hash[t]$
$d \leftarrow \mathsf{untag}(\mathsf{f}(t))$
if $b \wedge d \in range(Hash)$:
  throw *abort*
$Hash[t] \leftarrow d$
return d

Fig. 20: Game $\mathtt{Gacr}^{\mathsf{f},b}$.

Lemma A.2 (Agile Collision Resistance). *For all adversaries $\mathcal{A}$ and $\mathsf{f} \in \{\mathsf{hash}, \mathsf{xtr}, \mathsf{xpd}\}$ and a set $\mathcal{H}$ where each $alg \in \mathcal{H}$ has a different output length, we have that*

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gacr}^{\mathsf{f},b}) \leq \sum_{alg \in \mathcal{H}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{alg,\mathsf{f}}, \mathtt{Gcr}^{\mathsf{hash}\text{-}alg,b})$$

where $\mathcal{R}_{alg,\mathsf{f}} = \mathcal{R}_{alg} \to \mathcal{R}_{\mathsf{f}}$, $\mathcal{R}_{alg}$ for $alg \in \mathcal{H}$ is defined in Fig. 21 using an arbitrary order $<$ on $\mathcal{H}$, and $\mathcal{R}_{\mathsf{f}}$ is taken from Lemma A.1.

Proof. The proof of this lemma relies on $\mathcal{H}$ only containing hash-functions with different output length. As a result of the output length being different, conceptually, in the agile collision-resistance game $\mathtt{Gacr}^{\mathsf{f},b}$, we could instead maintain different tables $Hash_{alg}$, since the intersection of the ranges for different $alg \in \mathcal{H}$ is empty. The proof thus follows via hybrid argument over $alg \in \mathcal{H}$ using an arbitrary order $<$ on $\mathcal{H}$ and alg_{fst} being the smallest algorithm in $\mathcal{H}$ with respect to the order $<$ and alg_{lst} being the largest algorithm in $\mathcal{H}$ with respect to the order $<$. By construction of $\mathcal{R}_{alg}$, we have

$$\begin{aligned} \mathcal{R}_{alg_{\mathrm{fst}}} \to \mathtt{Gcr}^{alg_{\mathrm{fst}},0} &\overset{\text{code}}{\equiv} \mathtt{Gacr}^{\mathsf{f},0} \\ \mathcal{R}_{alg_{\mathrm{lst}}} \to \mathtt{Gcr}^{alg_{\mathrm{lst}},1} &\overset{\text{code}}{\equiv} \mathtt{Gacr}^{\mathsf{f},1} \end{aligned} \tag{1}$$

and for all $alg < alg_{\mathrm{lst}}$ with successor alg', we have

$$\mathcal{R}_{alg} \to \mathtt{Gcr}^{alg,1} \overset{\text{code}}{\equiv} \mathcal{R}_{alg'} \to \mathtt{Gcr}^{alg',0} \tag{2}$$

$\underline{\underline{\mathcal{R}_{alg}}}$
$\mathsf{HASH}(t)$

assert $\mathsf{alg}(t) \in \mathcal{H}$
if $Hash[t] \neq \bot$:
  return $Hash[t]$
if $alg(t) = alg$:
  return $\mathsf{HASH}(\mathsf{untag}(t))$
$d \leftarrow \mathsf{untag}(\mathsf{f}(t))$
if $d \in range(Hash)$
    $\wedge\, \mathsf{alg}(t) < alg$:
  throw *abort*
$Hash[t] \leftarrow d$
return d

Fig. 21: Reduction $\mathcal{R}_{alg}$.

We then use the telescopic sum argument and triangle inequality for a standard hybrid arguments:

$$
\begin{aligned}
&\left|\Pr\left[1 \leftarrow \mathcal{A} \to \mathtt{Gacr}^{\mathsf{f},0}\right] - \Pr\left[1 \leftarrow \mathcal{A} \to \mathtt{Gacr}^{\mathsf{f},0}\right]\right| \\
&\overset{(1)}{=} \left|\Pr\left[1 \leftarrow \mathcal{A} \to \mathcal{R}_{alg_{\mathrm{fst}}} \to \mathtt{Gcr}^{alg_{\mathrm{fst}},0}\right] - \Pr\left[1 \leftarrow \mathcal{A} \to \mathcal{R}_{alg_{\mathrm{lst}}} \to \mathtt{Gcr}^{alg_{\mathrm{lst}},1}\right]\right| \\
&\overset{(2)}{=} \left|\sum_{alg \in \mathcal{H}} \Pr\left[1 \leftarrow \mathcal{A} \to \mathcal{R}_{alg} \to \mathtt{Gcr}^{\mathsf{f}\text{-}alg,0}\right] - \Pr\left[1 \leftarrow \mathcal{A} \to \mathcal{R}_{alg} \to \mathtt{Gcr}^{\mathsf{f}\text{-}alg,1}\right]\right| \\
&\leq \sum_{alg \in \mathcal{H}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{alg}, \mathtt{Gcr}^{\mathsf{f}\text{-}alg,b}) \\
&\leq \sum_{alg \in \mathcal{H}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{alg} \to \mathcal{R}_{\mathsf{f}}, \mathtt{Gcr}^{\mathsf{hash}\text{-}alg,b})
\end{aligned}
$$

This concludes the proof of Lemma A.2.

A.2 Handles, `Key` Packages, Uniqueness `Logs`

We now turn to the `Key` and `Log` packages, see Fig. 22 for their formal description. As the `K` and `L` package, the `Key` package is an array for storing keys and their honesty and `Log` contains a table to notice collisions. In comparison to `L`, the `Log` package has more *patterns* as well as a novel *mapping* mechanism.

Patterns. Instead of the patterns A (apsk), F (full) and Z (zero), now, there are also a D (dishonest), and a R (rewarding) pattern. In the case of an R abort, the special symbol *win* is returned instead of *abort*. The patterns are written in a table in Fig. 22 rather than inlined as **if-then**-branches.

Mapping. The `Log` now also contains a *mapping* parameter *map* which affects UNQ as follows: If $map = \infty$, then for all collisions on dishonest keys, UNQ returns the *first* handle with this key value. If $map = 1$, only the first collision between two distinct handles h and h' such that (1) they have the same key value, (2) exactly one of the handles has level 0, is mapped, i.e., the second handle is mapped to the first. For each key value, such mapping is only done once when $map = 1$.

Jumping ahead, we note that $map = \infty$ will be used for DH mappings and $map = 1$ will be used for APSK mappings in the proof of Lemma C.2. In Lemma A.3, we prove that the simplified `K` and `L` packages are equivalent to the special cases of `Key` and `Log`.

Lemma A.3 (Simplified key package equivalence). *For any name n:*

$$\mathtt{K}^0_{n,0..d} \to \mathtt{L}^Z \overset{code}{\equiv} \mathtt{Key}^0_{n,0..d} \to \mathtt{Log}^Z_n \tag{3}$$

$$\mathtt{K}^1_{n,0..d} \to \mathtt{L}^F \overset{code}{\equiv} \mathtt{Key}^1_{n,0..d} \to \mathtt{Log}^F_n \tag{4}$$

$$\begin{matrix}\mathtt{K}^1_{n,0} \\ \mathtt{K}^0_{n,1..d}\end{matrix} \to \mathtt{L}^A \overset{code}{\equiv} \begin{matrix}\mathtt{Key}^1_{n,0} \\ \mathtt{Key}^0_{n,1..d}\end{matrix} \to \mathtt{Log}^A_n \tag{5}$$

$\underline{\underline{\mathsf{Key}^b_{n,\ell}}}$

$\underline{\mathsf{SET}_{n,\ell}(h, hon, k^\star)}$

assert $\mathsf{name}(h) = n$
assert $\mathsf{level}(h) = \ell$
assert $\mathsf{alg}(k^\star) = \mathsf{alg}(h)$
$k \leftarrow \mathsf{untag}(k^\star)$
assert $\mathsf{len}(h) = |k|$
if $\mathsf{Q}_n(h) \neq \bot$:
  return $\mathsf{Q}_n(h)$
if b :
  if hon :
    $k \leftarrow\!\!\$\ \{0,1\}^{\mathsf{len}(h)}$
$h' \leftarrow \mathsf{UNQ}_n(h, hon, k)$
if $h' \neq h$:
  return h'
$K_{n,\ell}[h] \leftarrow (k, hon)$
return h

$\underline{\mathsf{GET}_{n,\ell}(h)}$

assert $K_{n,\ell}[h] \neq \bot$
$(k^*, hon) \leftarrow K_{n,\ell}[h]$
$k \leftarrow \mathsf{tag}_h(k^*)$
return (k, hon)

$\underline{\underline{\mathsf{Log}^{P,map}_n}}$

$\underline{\mathsf{Q}_n(h)}$

if $Log_n[h] = \bot$:
  return $\bot$
else $(h', \_, \_) \leftarrow Log_n[h]$
  return h'

$\underline{\mathsf{UNQ}_n(h, hon, k)}$

if $(\exists\ h' : \mathrm{Log}_n[h'] = (h', hon', k)$
  $\wedge\ \mathsf{level}(h) = r \wedge \mathsf{level}(h^\star) = r')$:
  if $map(r, hon, r', hon' \mathrm{J}_n[k])$:
    $\mathrm{Log}_n[h] \leftarrow (h', hon, k)$
    $\mathrm{J}_n[k] \leftarrow 1$
    return h'
if $(\exists\ h^\star : \mathrm{Log}_n[h^\star] = (h', hon', k)$
  $\wedge\ \mathsf{level}(h) = r \wedge \mathsf{level}(h^\star) = r')$:
  $P(r, hon, r', hon')$
$Log_n[h] \leftarrow (h, hon, k)$
return h

P	the command $P(r, hon, r', hon')$ is
Z	$\emptyset$
A	**if** $hon = hon' = 0 \wedge r = r' = 0$: **throw** *abort*
D	**if** $hon = hon' = 0$: **throw** *abort*
R	**if** $hon = hon' = 0$: **throw** *abort* **else throw** *win*
F	**throw** *abort*

map	the command $map(r, hon, r', hon', \mathrm{J}_n[k])$ is
0	0
1	$hon = hon' = 0 \wedge$ $\wedge\ r \neq r' \wedge 0 \in \{r, r'\} \wedge \mathrm{J}_n[k] \neq 1$
∞	$hon = hon' = 0$

Fig. 22: Oracles that define the $\mathsf{Key}^b_{n,\ell}$ (left) and Log_n package (right). $\mathsf{SET}(h, hon, k)$ registers a key value k for handle h with honesty status hon. The key value and its honesty can be accessed via GET. When $map = \infty$, UNQ returns the first handle that registered the same key; when $map = 1$, UNQ returns the first handle, but only if this is the *first* collision between an application PSK (level $= 0$) and an application PSK (level > 0); see discussion before Lemma C.2. We omit to write map when $map = 0$. Collisions in unmapped keys can cause aborts as specified by the pattern $P \in \{Z, A, D, R, F\}$. Note that the asserts $\mathsf{name}(h) = psk$ and $\mathsf{level}(h) = 0$ ensure that $h = \mathsf{psk}\langle ctr, alg\rangle$.

Proof. We first start with the differences in the code of K and Key and then turn to the differences in the code of L and Log.

Code of Key The SET oracle of Key, as opposed to the SET oracle of K, uses Q to ensure consistent input-output behavior on a repeated handled h. If $map = 0$, then $\mathsf{UNQ}(h, *, *)$ always returns h (unless it aborts). Thus, for the cases considered in Eq. 3-5, using Q or K for this purpose is equivalent.

Similarly, the SET oracle of Key does not store hon and k in K if $h' \neq h$. However, this never happens in the cases Eq. 3-5, since $map = 0$ and thus, these lines are irrelevant for the behaviour of Key for the cases we consider.

Another difference is that the SET oracle of Key *always* queries UNQ and also when $b = 0$. This affects Eq. 3 and 5. However, in the case that the pattern is Z, the query has no effect (Eq. 3). And if the level is 1 or higher, then the query also has no effect when the pattern is A (Eq. 5).

Code of Log Since we compare cases where $map = 0$, the $map = 1$ and $map = \infty$ branches of Log are irrelevant. The different encoding of A is equivalent, and the patterns D, R and F do not appear in Eq. 3-5.

$\underline{\mathtt{Nkey}_n}$

$\mathsf{SET}_n(h, hon, k)$

assert $\mathsf{name}(h) = n$
if $\mathsf{Q}_n(h) \neq \perp$:
  return $\mathsf{Q}_n(h)$
$h' \leftarrow \mathsf{UNQ}_n(h, hon, k)$
if $h' \neq h$:
  return h'
$K_n[h] \leftarrow (k, hon)$
return h

$\mathsf{GET}_{n,0..d}(h)$

assert $K_n[h] \neq \perp$
$(k^*, hon) \leftarrow K_n[h]$
$k \leftarrow \mathsf{tag}_h(k^*)$
return (k, hon)

(a) Nkey

$\underline{\mathtt{Nkey}_{0salt}}$

State

K_{0salt} : key table
Initial state:
  $K_{0salt}[\mathsf{0salt}] \leftarrow 0$

$\underline{\mathtt{Nkey}_{0ikm}}$

State

K_{0ikm} : key table
Initial state:
for $alg \in \mathcal{H}$:
  $K_{0ikm}[\mathsf{0ikm}\langle alg\rangle] \leftarrow 0^{\mathsf{len}(alg)}$

$\underline{\mathtt{Nkey}_{dh}}$

State

K_{dh} : key table
Initial state:
for $alg \in \mathcal{H}$:
  $K_{dh}[\mathsf{noDH}\langle alg\rangle] \leftarrow 0^{\mathsf{len}(alg)}$

$\underline{\mathtt{Key}_{psk,0}}$

State

$K_{psk,0}$: key table
Initial state:
for $alg \in \mathcal{H}$:
  $K_{0psk,0}[\mathsf{noPSK}\langle alg\rangle] \leftarrow 0^{\mathsf{len}(alg)}$

(b) Initial State

Fig. 23: Nkey and state initialization

`Nkey` For Diffie-Hellman keys, 0salt, and 0ikm, we use package indices that have no level, since keys need to be available on *all* levels. Uniqueness proofs would suffer from introducing unnecessary redundancy and thus, we introduce a `Nkey` package which stores the relevant keys *globally* and returns them without considering the level from which they were requested. That is, `Nkey` exposes a SET_n oracle without a level and $\mathsf{GET}_{n,\ell}$ oracles for all levels $\ell \in \{0, .., d\}$. Note that some of the handles and keys miss attributes commonly stored in the tags of keys and handles, and thus, the SET_n oracle of `Nkey` performs less checks on them.

In particular, we use the packages $\texttt{Nkey}_{0salt}$, $\texttt{Nkey}_{dh}$ and $\texttt{Nkey}_{0ikm}$. We provide the code of $\texttt{Nkey}_n$ in Figure 23a. We initialize the state of the $\texttt{Nkey}_n$ packages as specified in Figure 23b. For $n \in \{0salt, 0ikm\}$, this is necessary, since we do not expose a SET_n query to the adversary and since these packages only store fixed zero-values. These zero-values corresponds to setting missing values to a zero string which is of the same length as the output of *alg*, as specified by the TLS RFC. Note that for *0ikm*, we store a 0-value for each algorithm *alg* of the length which corresponds to the algorithm, but for *0salt*, we only store a single 1-bit-long 0-key. This is because hmac (see Fig. 19) pads keys with zeroes up to their block length and thus, storing multiple zero values would introduce redundancy in the model without a correspondence in real-life.

Equivalence As the `K` and `Key` package, the `Nkey` packages are code-equivalent to the `Nk` packages introduced in Section 4.5 when composed with a zero log.

Lemma A.4 (Simplified `Nkey` package equivalence). *For any name n:*

$$\texttt{Nk} \rightarrow \texttt{L}^Z \overset{code}{\equiv} \texttt{Nkey} \rightarrow \texttt{Log}^Z$$

Proof. The SET oracle of `Nkey`, as opposed to the SET oracle of `Nk`, uses Q to ensure consistent input-output behavior on a repeated handled h. since $map = 0$, the $\mathsf{UNQ}(h, *, *)$ always returns h (unless it aborts). Thus, as we considerd $P = Z$, using Q or K for this purpose is equivalent.

A.3 Pseudorandomness Assumptions

We now turn to our pseudorandomness assumptions, all depicted in Fig. 24. All of the games are structured as sketched in Section 2, i.e., an ideal (upper) $\texttt{Key}^1$ package stores honest and random input keys (as well as dishonest, concrete keys which are chosen by the adversary), then some package retrieves these keys via GET and stores the result in a lower $\texttt{Key}^b$ package. If $b = 0$, the concrete keys are stored. If $b = 1$ and the handle is honest, then a uniformly random key is drawn by the lower $\texttt{Key}^1$ package instead. The adversary can retrieve keys via GET to the lower $\texttt{Key}^b$ package. We add a $\texttt{HASH}^1$ package for performing hash-computation—that package is already idealized, i.e., collision-free as to avoid collisions on input so that these are exclusively pseudorandomness assumptions.

Firstly, we assume that xpd, when keyed with a uniformly random key, behaves like a pseudorandom function. See Fig. 24a for the corresponding security

game. Since the computation of the *psk* requires to increase the level, we formulate a separate game $\texttt{Gxpd}^b_{psk,\ell}$ in Fig. 24c. We also formulate a separate pseudorandonmness game $\texttt{Gxpd}^b_{esalt,\ell}$ in Fig. 24b for the derivation of *esalt* and its siblings, since the pattern on the $\texttt{Log}^R_{esalt}$ package is R when we apply the pseudorandomness assumption for *esalt* and its siblings.

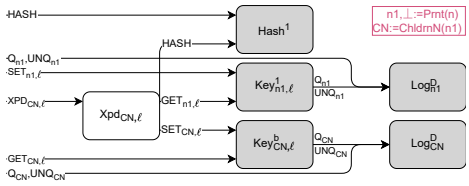
Analogously to the assumptions for xpd, we formulate three assumptions on xtr, one for each of the three extraction operations. Namely, we assume that xtr, when given a uniformly random k_{psk} (as second input) and a fixed k_{0salt} (as first input) returns a pseudorandom key—this could be a statistical extractor assumption, but since this property has not been proven statistically, we formulate it as a computational assumption. Dually, we assume that xtr, when given a uniformly random k_{hsalt} (as first input) and a fixed k_{0ikm} (as second input) returns a pseudorandom key. This property could also hold statistically, and its computational variant is implied by the pseudorandom function properties of hmac, on which xtr is built [48,8,43]. The corresponding games are stated in Fig. 24d and Fig. 24f. The assumption on xtr in Fig. 24e is a special case as it contains an $\texttt{Xtr}^1$ package (and an $\texttt{Log}^R$ package). The importance of $\texttt{Xtr}^1$ is that it samples a random key if the Diffie-Hellman secret is honest. Therefore, output key values whose handles correspond to derivations based on *honest* Diffie-Hellman secrets do not depend on their inputs, regardless of whether the bit in the lower $\texttt{Key}^b_{hs}$ package is 0 or 1. For all other keys, the game models pseudorandomness based on the *esalt* honesty using the same argument as the other xtr assumptions.

Salted Oracle Diffie-Hellman The game $\texttt{Gsodh}^b$ in Fig. 24g formalizes the SODH assumption introduced in Section 3.3 in terms of the Xtr, DHKey and DH packages (See Fig. 12): If the bit b in the $\texttt{Xtr}_{hs}$ is set to 1 and the Diff-Hellman key (in DHKey) is honest, the output key is sampled at random.

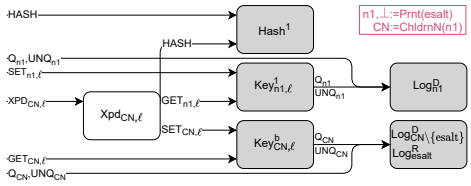
To make this assumption meaningful (i.e., not trivially wrong), it is crucial for the assumption that the pattern on $\texttt{LOG}_{esalt}$ ensures that all keys are unique—regardless of whether they are honest or dishonest—since all of these keys are set by the adversary, and a colliding key pair would allow for a trivial attack, since the concrete computation on these keys would be the same whereas two randomly sampled keys are highly likely to be different.

Pre-image resistance We now turn to pre-image resistance, i.e., it should be hard for an adversary to find a pre-image of a uniformly random string. We need pre-image resistance in two forms modeled as $\texttt{Gpi}^P_{O^*}$ (Fig. 24h) and $\texttt{Gpi}^P_{esalt}$ (Fig. 24i). The first allows us to require full (pattern F) uniqueness for output keys which requires that honestly generated output keys are collision-free (as opposed to the D pattern that only restricts collisions between two dishonest keys). Pre-image resistance as modeled in $\texttt{Gpi}^P_{esalt}$ is used to replace the R pattern (where the adversary directly wins if it can cause an collision with an honest key, see Lemma D.4) with an D pattern by observing that it is hard for an adversary to cause the winning condition.

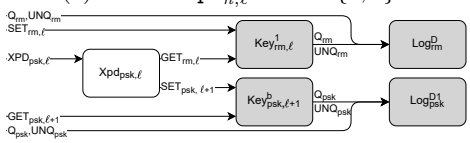
Definition A.5 (Advantages). *For a set N and* PrntN *with $XPN \subseteq N$ and $n \in XPN$, we define the* xpd *pseudorandomness advantages*



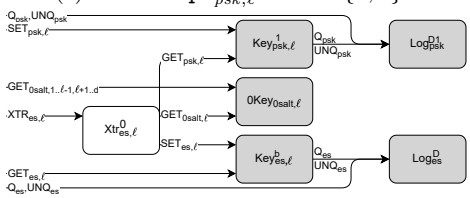
(a) Game $\texttt{Gxpd}^b_{n,\ell}$ for $b \in \{0,1\}$



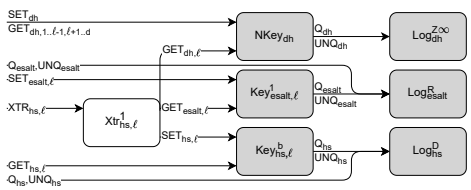
(b) Game $\texttt{Gxpd}^b_{esalt,\ell}$ for $b \in \{0,1\}$



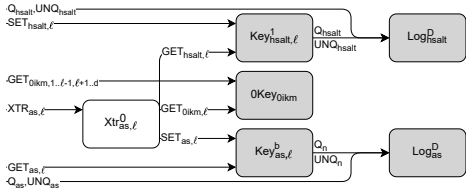
(c) Game $\texttt{Gxpd}^b_{psk,\ell}$ for $b \in \{0,1\}$



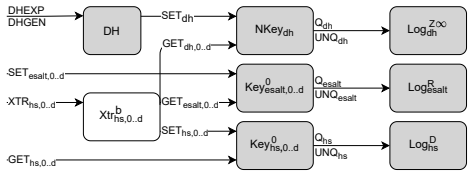
(d) Game $\texttt{Gxtr1}^b_{es,\ell}$ for $b \in \{0,1\}$



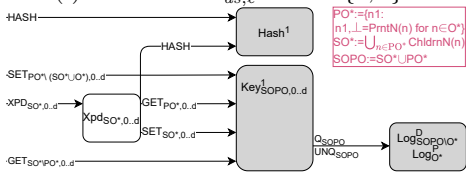
(e) Game $\texttt{Gxtr2}^b_{hs,\ell}$ for $b \in \{0,1\}$



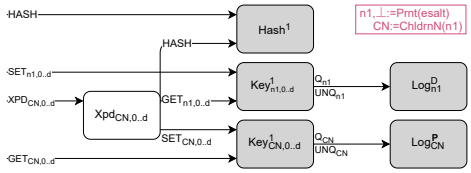
(f) Game $\texttt{Gxtr3}^b_{as,\ell}$ for $b \in \{0,1\}$



(g) Game $\texttt{Gsodh}^b$ for $b \in \{0,1\}$



(h) Games $\texttt{Gpi}^P_{O*}$ for $P \in \{D, F\}$



(i) Games $\texttt{Gpi}^P_{esalt}$ for $P \in \{R, D\}$

Fig. 24: Pseudorandomness and pre-image-resistance games

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gxpd}^0_{n,\ell}, \mathtt{Gxpd}^1_{n,\ell}) := \left|\Pr\left[1 = \mathcal{A} \to \mathtt{Gxpd}^0_{n,\ell}\right] - \Pr\left[1 = \mathcal{A} \to \mathtt{Gxpd}^1_{n,\ell}\right]\right|$$

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gxpd}^0_{psk,\ell}, \mathtt{Gxpd}^1_{psk,\ell}) := \left|\Pr\left[1 = \mathcal{A} \to \mathtt{Gxpd}^0_{psk,\ell}\right] - \Pr\left[1 = \mathcal{A} \to \mathtt{Gxpd}^1_{psk,\ell}\right]\right|$$

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gxpd}^0_{esalt,\ell}, \mathtt{Gxpd}^1_{esalt,\ell}) := \left|\Pr\left[1 = \mathcal{A} \to \mathtt{Gxpd}^0_{esalt,\ell}\right] - \Pr\left[1 = \mathcal{A} \to \mathtt{Gxpd}^1_{esalt,\ell}\right]\right|.$$

Additionally, we define the xtr *pseudorandomness advantages*

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gxtr1}^0_{es,\ell}, \mathtt{Gxtr1}^1_{es,\ell}) := \left|\Pr\left[1 = \mathcal{A} \to \mathtt{Gxtr1}^0_{es,\ell}\right] - \Pr\left[1 = \mathcal{A} \to \mathtt{Gxtr1}^1_{es,\ell}\right]\right|$$

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gxtr2}^0_{hs,\ell}, \mathtt{Gxtr2}^1_{hs,\ell}) := \left|\Pr\left[1 = \mathcal{A} \to \mathtt{Gxtr2}^0_{hs,\ell}\right] - \Pr\left[1 = \mathcal{A} \to \mathtt{Gxtr2}^1_{hs,\ell}\right]\right|$$

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gxtr3}^0_{as,\ell}, \mathtt{Gxtr3}^1_{as,\ell}) := \left|\Pr\left[1 = \mathcal{A} \to \mathtt{Gxtr3}^0_{as,\ell}\right] - \Pr\left[1 = \mathcal{A} \to \mathtt{Gxtr3}^1_{as,\ell}\right]\right|.$$

The SODH advantage is defined as

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gsodh}^0, \mathtt{Gsodh}^1) := \left|\Pr\left[1 = \mathcal{A} \to \mathtt{Gsodh}^0\right] - \Pr\left[1 = \mathcal{A} \to \mathtt{Gsodh}^1\right]\right|.$$

Finally, we define the pre-image resistance advantages as

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gpi}^D_{O^*}, \mathtt{Gpi}^F_{O^*}) := \left|\Pr\left[1 = \mathcal{A} \to \mathtt{Gpi}^D_{O^*}\right] - \Pr\left[1 = \mathcal{A} \to \mathtt{Gpi}^F_{O^*}\right]\right|$$

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gpi}^R_{esalt}, \mathtt{Gpi}^D_{esalt}) := \left|\Pr\left[1 = \mathcal{A} \to \mathtt{Gpi}^R_{esalt}\right] - \Pr\left[1 = \mathcal{A} \to \mathtt{Gpi}^D_{esalt}\right]\right|$$

For a pair of games $\mathtt{G}^0$, $\mathtt{G}^1$, *we use the abbreviation* $\mathsf{Adv}(\mathcal{A}, \mathtt{G}^0, \mathtt{G}^1) := \mathsf{Adv}(\mathcal{A}, \mathtt{G}^b)$.

B Key Schedule Theorem

In this appendix, we show that Theorem 4.5 follows from Theorem C.1 (proved in Appendix C) and Lemma E.1, Lemma E.6 and Lemma E.9 (proved in Appendix E). We start by re-stating Theorem 4.5.

Recall that *XPR* is the set of output key name representatives of `Xpd` packages. The need for such representatives arise from us indexing both `Xtr` packages and `Xpd` packages by output keys. As *psk* is the main root of the key schedule it is natural to pick it as a representative; as *esalt* plays a special role in our proof due to SODH, it is also a natural choice. The other representatives are picked arbitrarily among siblings.

Theorem 4.5. *Let ks be a TLS-like key schedule with representative set XPR and a set of separation points S. Let* $d \in \mathbb{N}$. *Let* $PO^* := \{n : \exists n' \in O^* : (n, \_) = \mathsf{PrntN}(n')\}$ *and* $SO^* := \bigcup\{\mathsf{ChldrnN}(n_1) \mid n_1 \in PO^*\}$. *There exists an efficient simulator* $\mathcal{S}$ *such that for all adversaries* $\mathcal{A}$ *which make queries for at most d resumption levels, use at most* $s_{n,\ell,alg} = t^{hon=1}_{n,alg}$ *honest and* $t^{hon=0}_{n,alg}$ *dishonest parent keys for algorithm alg to generate keys with name n at level* ℓ, *and let* $t_{n,alg}$ *be* $\max\{t^{hon=0}_{n,alg}, t^{hon=1}_{n,alg}\}$.

$$
\begin{aligned}
&\mathsf{Adv}(\mathcal{A}, \mathtt{Gks}^0, \mathtt{Gks}^1(\mathcal{S})) \leq \\
&\sum_{alg\in\mathcal{H}} \Big(\mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{main}_{cr} \to \mathcal{R}_{alg,\mathsf{hash}}, \mathtt{Gcr}^{\mathsf{hash}\text{-}alg,b}) \\
&+ \sum_{j\in\{Z,D\},\mathsf{f}\in\{\mathsf{xtr},\mathsf{xpd}\}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{main}_{j} \to \mathcal{R}_{alg,\mathsf{f}}, \mathtt{Gcr}^{\mathsf{hash}\text{-}alg,b}) \Big) \\
&+ \max_{i\in\{0,1\}} \Big[\sum_{grp\in\mathcal{G}} \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{sodh} \to \mathcal{R}^{grp}_{sodh2}, \mathtt{Gsodh2}^{grp,b}) \\
&+ 2 \cdot \min_{alg\in\mathcal{H}} \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{sodh} \to \mathcal{R}^{alg}_{sodh\text{-}cr}, \mathtt{Gcr}^{\mathsf{hash}\text{-}alg,b}) \\
&+ \sum_{alg\in\mathcal{H}} \Big(\sum_{0\leq\ell\leq d} \big[s_{es,\ell,alg} \cdot \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{es,\ell,alg}, \mathtt{Gpr}^{\mathsf{xtr}\text{-}alg^\dagger,b}) \\
&+ s_{hs,\ell,alg} \cdot \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{hs,\ell,alg}, \mathtt{Gpr}^{\mathsf{xtr}\text{-}alg,b}) \\
&+ s_{as,\ell,alg} \cdot \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{as,\ell,alg}, \mathtt{Gpr}^{\mathsf{xtr}\text{-}alg,b}) \\
&+ \sum_{n\in XPR} s_{n,\ell,alg} \cdot \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{n,\ell,alg}, \mathtt{Gpr}^{\mathsf{xpd}\text{-}alg,b}) \big] \\
&+ 2 \cdot \Big[\Big(\sum_{0\leq\ell\leq d} s_{esalt,\ell,alg} \Big) \cdot \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{esalt,pi,alg}, \mathtt{Gpr}^{\mathsf{xpd}\text{-}alg,b}) \\
&+ \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{esalt,pi\text{-}cr,alg}, \mathtt{Gcr}^{\mathsf{hash}\text{-}alg,b}) \Big] \\
&+ \sum_{n\in SO^*\cap XPR} 2 \cdot \Big[\Big(\sum_{0\leq\ell\leq d} s_{n,\ell,alg} \Big) \cdot \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{n,pi,alg}, \mathtt{Gpr}^{\mathsf{xpd}\text{-}alg,b}) \\
&+ \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{n,pi\text{-}cr,alg}, \mathtt{Gcr}^{\mathsf{hash}\text{-}alg,b}) \Big] \\
&+ 2^{-\mathsf{len}(alg)} \cdot \Big[(c+1) \cdot t^2_{esalt,alg} + (2c+6) \cdot t^2_{es,alg} + \sum_{n\in PO*} ((2c+6) \cdot t^2_{n,alg}) \Big] \Big) \Big],
\end{aligned}
$$

$\mathcal{A}_i$ defined in Appendix D.1, modifies $\mathcal{A}$ without significantly changing its complexity: it behaves as $\mathcal{A}$ except that it returns i on some aborts; $\mathcal{R}^{main}_{*} := \mathcal{R}^{ch\text{-}map} \to \mathcal{R}_*$ when replacing $*$ by cr, Z, D or sodh; $\mathcal{R}^{main}_{*,alg} := \mathcal{R}^{ch\text{-}map} \to \mathcal{R}^{alg}_{*}$ when replacing $*$ by n, pi, n, pi-cr, esalt, pi or esalt, pi-cr; $\mathcal{R}^{main}_{*,\ell,alg} := \mathcal{R}^{ch\text{-}map} \to \mathcal{R}_{*,\ell} \to \mathcal{R}^{alg}_{*,\ell}$ when replacing $*$ by es, hs, as, n; the simulator $\mathcal{S}$ is marked in grey in Fig. 26b; $\mathcal{R}_{alg,\mathsf{f}}$ is defined in Lemma A.2, $\mathcal{R}_{sodh}$ is defined in Fig. 32a, $\mathcal{R}_{es,\ell}$ is defined in Fig. 34a, $\mathcal{R}_{hs,\ell}$ and $\mathcal{R}_{as,\ell}$ are defined analogously, and $\mathcal{R}_{n,\ell}$ for $n \in XPR$ and $0 \le \ell \le d$ are defined in Fig. 34b; $\mathcal{R}^{alg}_{n,\ell}$ for $n \in XPR \cup \{es, hs, as\}$ are defined in Lemma E.6; $\mathcal{R}^{alg}_{sodh\text{-}cr}$ and $\mathcal{R}^{grp}_{sodh2}$ are defined in Lemma E.1, $\mathcal{R}_{esalt,pi}$ is defined in Fig. 32c and $\mathcal{R}_{n\in PO^*,pi}$ is defined in Fig. 32d; c is a small constant which depends on the min-entropy of the distribution $\mathsf{xtr}(k, U_{\mathsf{len}(alg)})$, where k is a fixed key and $U_{2^{\mathsf{len}(alg)}}$ denotes the uniform distribution of strings of length $\mathsf{len}(alg)$.

The proof of Theorem 4.5 follows from first reducing the security of the key schedule to modular assumptions (Theorem C.1) and then, in Appendix E, reducing the modular assumptions (defined in Appendix A) to their monolithic counterparts (defined in Section 3). We now first state Theorem C.1 and Lemma E.1, Lemma E.6 and Lemma E.9 and then show that they imply Theorem 4.5.

Theorem C.1. *Let ks be a TLS-like key schedule with representative set XPR and separation points S. Let $d \in \mathbb{N}$. There is an efficient simulator $\mathcal{S}$ such that for all adversaries $\mathcal{A}$ which make queries for at most d resumption levels,*

$$
\begin{aligned}
&\mathsf{Adv}(\mathcal{A}, \mathtt{Gks}^0, \mathtt{Gks}^1(\mathcal{S})) \\
&\le \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{main}_{cr}, \mathtt{Gacr}^{\mathsf{hash},b}) \\
&+ \sum_{j\in\{Z,D\},\mathsf{f}\in\{\mathsf{xtr},\mathsf{xpd}\}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{main}_{j,\mathsf{f}}, \mathtt{Gacr}^{\mathsf{hash},b}) \\
&\quad + \max_{i\in\{0,1\}} \big[\mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{sodh}, \mathtt{Gsodh}^b) \\
&\qquad \sum_{\ell=0}^{d-1} \big(\mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{es,\ell}, \mathtt{Gxtr}^b_{es,\ell}) \\
&\qquad\quad + \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{hs,\ell}, \mathtt{Gxtr}^b_{hs,\ell}) \\
&\qquad\quad + \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{as,\ell}, \mathtt{Gxtr}^b_{as,\ell}) \\
&\qquad + \sum_{n\in XPR} \big(\mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{n,\ell}, \mathtt{Gxpd}^b_{n,\ell})\big)\big) \\
&\qquad\quad + \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{esalt,pi}, \mathtt{Gpi}^b_{esalt}) \\
&\qquad\quad + \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{O^*,pi}, \mathtt{Gpi}^b_{O^*})\big],
\end{aligned}
$$

where $\mathcal{A}_i$ defined in Appendix D.1, modifies $\mathcal{A}$ without significantly changing its complexity: it behaves as $\mathcal{A}$ except that it returns i on some aborts; where

$\mathcal{R}_*^{main} := \mathcal{R}^{ch\text{-}map} \to \mathcal{R}_*$ *when replacing* $*$ *by cr,* (Z,f), (D,f), *sodh, es, hs, as, n,* O^*, pi *or* $esalt, pi$, *the simulator* $\mathcal{S}$ *is marked in grey in Fig. 26b,* $\mathcal{R}_{sodh}$ *is defined in Fig. 32a,* $\mathcal{R}_{es,\ell}$ *is defined in Fig. 34a,* $\mathcal{R}_{hs,\ell}$ *and* $\mathcal{R}_{as,\ell}$ *are defined analogously, and* $\mathcal{R}_{n,\ell}$ *for* $n \in XPR$ *and* $0 \le \ell \le d$ *is defined in Fig. 34b,* $\mathcal{R}_{esalt,pi}$ *is defined in Fig. 32c and* $\mathcal{R}_{O^*,pi}$ *is defined in Fig. 32d.*

Lemma E.1 (Salted-ODH Advantage). *For all* $\mathcal{A}$, *it holds that*

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gsodh}^0, \mathtt{Gsodh}^1) \le 2 \cdot \min_{alg \in \mathcal{H}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{alg}_{sodh\text{-}cr}, \mathtt{Gcr}^{\mathsf{hash}\text{-}alg,b}) + \sum_{grp \in \mathcal{G}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{grp}_{sodh}, \mathtt{Gsodh2}^{grp,b}),$$

where $\mathcal{R}^{alg}_{sodh\text{-}cr} := \mathcal{R}_{sodh4} \to \mathcal{R}^{alg}_{sodh2\text{-}cr,alg}$ *and* $\mathcal{R}^{grp}_{sodh} := \mathcal{R}_{sodh4} \to \mathcal{R}_{sodh2} \to \mathcal{R}_{grp}$. *We define the reductions* $\mathcal{R}_{sodh4}$ *and* $\mathcal{R}_{grp}$ *in Lemma E.2 and Lemma E.4, respectively, and we describe* $\mathcal{R}^{alg}_{sodh2\text{-}cr,alg}$ *and* $\mathcal{R}_{sodh2}$ *in Lemma E.3.*

Lemma E.6 (Advantages). *Let* $n \in XPR$ *and let* $(n_1, \_) = \mathsf{PrntN}(n)$. *Let* $\mathcal{A}$ *be an adversary that generates at most* $s_{n,\ell,alg}$ *honest keys for algorithm alg with name* n_1 *at level* ℓ *via* $\mathsf{SET}_{n_1,\ell}(*,1,*)$ *queries. The* xpd *pseudorandomness advantage is bounded by*

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gxpd}^0_{n,\ell}, \mathtt{Gxpd}^1_{n,\ell}) \le \sum_{alg \in \mathcal{H}} s_{n,\ell,alg} \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{alg}_{n,\ell}, \mathtt{Gpr}^{\mathsf{xpd}\text{-}alg,b}) \tag{6}$$

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gxpd}^0_{psk,\ell}, \mathtt{Gxpd}^1_{psk,\ell}) \le \sum_{alg \in \mathcal{H}} s_{psk,\ell,alg} \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{alg}_{psk,\ell}, \mathtt{Gpr}^{\mathsf{xpd}\text{-}alg,b}) \tag{7}$$

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gxpd}^0_{esalt,\ell}, \mathtt{Gxpd}^1_{esalt,\ell}) \le \sum_{alg \in \mathcal{H}} s_{esalt,\ell,alg} \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{alg}_{esalt,\ell}, \mathtt{Gpr}^{\mathsf{xpd}\text{-}alg,b}) \tag{8}$$

Let $\mathcal{A}$ *be an adversary that generates at most* $s_{es,\ell,alg}$ *honest keys for algorithm alg with name psk at level* ℓ *via* $\mathsf{SET}_{psk,\ell}(*,1,*)$ *queries. The* xtr *pseudorandomness advantage is bounded by*

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gxtr1}^0_{es,\ell}, \mathtt{Gxtr1}^1_{es,\ell}) \le \sum_{alg \in \mathcal{H}} s_{es,\ell,alg} \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{alg}_{es,\ell}, \mathtt{Gpr}^{\mathsf{xtr}^\dagger\text{-}alg,b}) \tag{9}$$

Let $\mathcal{A}$ *be an adversary that generates at most* $s_{es,\ell,alg}$ *honest keys for algorithm alg with name esalt at level* ℓ *via* $\mathsf{SET}_{esalt,\ell}(*,1,*)$ *queries. Let* n_{alg} *is*

an upper bound on the sum of UNQ_{esalt} *and* $\mathsf{SET}_{esalt,\ell}$ *queries made by* $\mathcal{A}$*, and let* c *be a small constant which depends on the min-entropy of the distribution* $f(k, U_{\mathsf{len}(alg)})$*, where* k *is a fixed key and* $U_{2^{\mathsf{len}(alg)}}$ *denotes the uniform distribution of strings of length* $\mathsf{len}(alg)$*. The* xtr *pseudorandomness advantage is bounded by*

$$\begin{aligned}&\mathsf{Adv}(\mathcal{A}, \mathtt{Gxtr2}^0_{hs,\ell}, \mathtt{Gxtr2}^1_{hs,\ell})\\ \leq \sum_{alg\in\mathcal{H}} s_{hs,\ell,alg}&\cdot\mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{alg}_{hs,\ell}, \mathtt{Gpr}^{\mathsf{xtr}\text{-}alg,b}) + (c+1)\cdot\frac{n^2_{alg}}{2^{\mathsf{len}(alg)}} \qquad (10)\end{aligned}$$

Let $\mathcal{A}$ *be an adversary that generates at most* $s_{as,\ell,alg}$ *honest keys for algorithm* alg *with name* $hsalt$ *at level* ℓ *via* $\mathsf{SET}_{hsalt,\ell}(*,1,*)$ *queries. The* xtr *pseudorandomness advantage is bounded by*

$$\begin{aligned}&\mathsf{Adv}(\mathcal{A}, \mathtt{Gxtr3}^0_{as,\ell}, \mathtt{Gxtr3}^1_{as,\ell})\\ \leq \sum_{alg\in\mathcal{H}} s_{as,\ell,alg}&\cdot\mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{alg}_{as,\ell}, \mathtt{Gpr}^{\mathsf{xtr}\text{-}alg,b}) \qquad (11)\end{aligned}$$

The reductions are defined as follows:

$$\begin{aligned}
\mathcal{R}^{alg}_{n,\ell} &:= \mathcal{R}^{rmprio,\mathsf{xpd}}_{n,\ell} \to \mathcal{R}^{\mathsf{f}\text{-}alg}_{\mathtt{G\text{-}M\text{-}Pr\text{-}io}}\ ,\ \textit{where}\ f = \mathsf{xpd}\\
\mathcal{R}^{alg}_{psk,\ell} &:= \mathcal{R}^{rmprio,\mathsf{xpd}}_{psk,\ell} \to \mathcal{R}^{\mathsf{f}\text{-}alg}_{\mathtt{G\text{-}M\text{-}Pr\text{-}io}}\ ,\ \textit{where}\ f = \mathsf{xpd}\\
\mathcal{R}^{alg}_{esalt,\ell} &:= \mathcal{R}^{rmprio,\mathsf{xpd}}_{esalt,\ell} \to \mathcal{R}^{\mathsf{f}\text{-}alg}_{\mathtt{G\text{-}M\text{-}Pr\text{-}io}}\ ,\ \textit{where}\ f = \mathsf{xpd}\\
\mathcal{R}^{alg}_{es,\ell} &:= \mathcal{R}^{rmprio,\mathsf{xtr}}_{es,\ell} \to \mathcal{R}^{\mathsf{f}\text{-}alg}_{\mathtt{G\text{-}M\text{-}Pr\text{-}io}}\ ,\ \textit{where}\ f = \mathsf{xtr}^{\dagger}\\
\mathcal{R}^{alg}_{hs,\ell} &:= \mathcal{R}^{rmprio,\mathsf{xtr}}_{hs,\ell} \to \mathcal{R}^{\mathsf{f}\text{-}alg}_{\mathtt{G\text{-}M\text{-}Pr\text{-}io}}\ ,\ \textit{where}\ f = \mathsf{xtr}\\
\mathcal{R}^{alg}_{as,\ell} &:= \mathcal{R}^{rmprio,\mathsf{xtr}}_{as,\ell} \to \mathcal{R}^{\mathsf{f}\text{-}alg}_{\mathtt{G\text{-}M\text{-}Pr\text{-}io}}\ ,\ \textit{where}\ f = \mathsf{xtr}
\end{aligned}$$

For $\mathcal{R}^{\mathsf{f}\text{-}alg}_{\mathtt{G\text{-}M\text{-}Pr\text{-}io}}$*, see Lemma E.5. Reduction* $\mathcal{R}^{rmprio,\mathsf{xpd}}_{n,\ell}$ *is defined in Fig. 54 (code) and Fig. 50b (graph),* $\mathcal{R}^{rmprio,\mathsf{xpd}}_{psk,\ell}$ *is defined in Fig. 55 (code) and Fig. 50f (graph),* $\mathcal{R}^{rmprio,\mathsf{xpd}}_{esalt,\ell}$ *is defined in Fig. 54 (code) and Fig. 50d (graph),* $\mathcal{R}^{rmprio,\mathsf{xtr}}_{es,\ell}$ *is defined in Fig. 51 (code) and Fig. 49b (graph),* $\mathcal{R}^{rmfprio,\mathsf{xtr}}_{hs,\ell}$ *is defined in Fig. 52 (code) and Fig. 49d (graph), and* $\mathcal{R}^{rmprio,\mathsf{xtr}}_{as,\ell}$ *is defined in Fig. 53 (code) and Fig. 49f (graph).*

Lemma E.9. *Let $\mathcal{A}$ be an adversary that generates at most $t^{hon=1}_{es,alg}$ honest and $t^{hon=0}_{es,alg}$ dishonest es keys, and let $t_{es,alg}$ be $\max\{t^{hon=0}_{es,alg}, t^{hon=1}_{es,alg}\}$. We have*

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gpi}^{R}_{esalt}, \mathtt{Gpi}^{D}_{esalt}) \tag{12}$$

$$\leq \sum_{alg\in\mathcal{H}} 2 \cdot \Bigg[s_{esalt,alg} \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{alg}_{esalt,pi}, \mathtt{Gpr}^{\mathsf{xpd}\text{-}alg,b}) + \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{alg}_{esalt,pi\text{-}cr}, \mathtt{Gcr}^{\mathsf{hash}\text{-}alg,b}) + (c+3) \cdot \frac{t^2_{es,alg}}{2^{\mathsf{len}(alg)}} \Bigg],$$

where $\mathcal{R}^{alg}_{esalt,pi} := \mathcal{R}^{PIE} \to \mathcal{R}^{mprio,xpd}_{n,0} \to \mathcal{R}^{\mathsf{xpd}\text{-}alg,D,\mathbf{D}/\mathbf{R}}_{\mathtt{G\text{-}M\text{-}Pr\text{-}io}}$ *and*

$\mathcal{R}^{alg}_{esalt,pi\text{-}cr} := \mathcal{R}^{PIE} \to \mathcal{R}^{mprio,xpd}_{n,0} \to \mathcal{R}^{cr,\mathsf{xpd},\mathbf{D}/\mathbf{R}} \to \mathcal{R}_{alg,\mathsf{xpd}}$,

c is a small constant which depends on the min-entropy of xpd *on random inputs and $\mathbf{R}$ is an abbreviation for $\mathbf{R} : \mathsf{ChldrnN}(es) \to \{R, D\}$ being defined as R on esalt and D everywhere else.*

Let the parent set of O^* be defined as $PO^* := \{n_1 : \exists n \in O^* : (n_1, \_) = \mathsf{PrntN}(n)\}$ and the sibling set of O^* as $SO^* := \bigcup_{n_1 \in PO^*} \mathsf{ChldrnN}(n_1)$. Let $\mathcal{A}$ be an adversary such that for each n in the representative set $SO^* \cap XPR$, $\mathcal{A}$ generates at most $s_{n,alg} = t^{hon=1}_{n,alg}$ honest and $t^{hon=0}_{n,alg}$ dishonest n keys for n_1 with $(n_1, \_) = \mathsf{PrntN}(n)$, and let $t_{n,alg}$ be $\max\{t^{hon=0}_{n,alg}, t^{hon=1}_{n,alg}\}$. We have

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gpi}^{D}_{O^*}, \mathtt{Gpi}^{F}_{O^*}) \tag{13}$$

$$\leq \sum_{alg\in\mathcal{H}, n\in SO^*\cap XPR} 2 \cdot \Bigg[s_{n,alg} \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{alg}_{n,pi}, \mathtt{Gpr}^{\mathsf{xpd}\text{-}alg,b}) + \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{alg}_{n,pi\text{-}cr}, \mathtt{Gcr}^{\mathsf{hash}\text{-}alg,b}) + (c+3) \cdot \frac{t^2_{n,alg}}{2^{\mathsf{len}(alg)}} \Bigg],$$

where $\mathcal{R}^{alg}_{n,pi} := \mathcal{R}^{PIO}_{n} \to \mathcal{R}^{mprio,xpd}_{n,0} \to \mathcal{R}^{\mathsf{xpd}\text{-}alg,D,\mathbf{D}/\mathbf{F}}_{\mathtt{G\text{-}M\text{-}Pr\text{-}io}}$ *and*

$\mathcal{R}^{alg}_{n,pi\text{-}cr} := \mathcal{R}^{PIO}_{n} \to \mathcal{R}^{mprio,xpd}_{n,0} \to \mathcal{R}^{cr,\mathsf{xpd},\mathbf{D}/\mathbf{F}} \to \mathcal{R}_{alg,\mathsf{xpd}}$,

c is a small constant which depends on the min-entropy of xpd *on random inputs, and $\mathbf{F} : \mathsf{ChldrnN}(n_1) \to \{D, F\}$ is F on the intersection of O^* and $\mathsf{ChldrnN}(n_1)$ and D, else.*

We now show that Theorem 4.5 follows from Theorem C.1, Lemma E.1, Lemma E.6 and Lemma E.9.

$$\begin{aligned} &\mathsf{Adv}(\mathcal{A}, \mathtt{Gks}^0, \mathtt{Gks}^1(\mathcal{S})) \\ &\overset{\text{Theorem C.1}}{\leq} \\ &\mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\text{main}}_{\text{cr}}, \mathtt{Gacr}^{\mathsf{hash},b}) \\ &+ \sum_{j\in\{Z,D\}, \mathsf{f}\in\{\mathsf{xtr},\mathsf{xpd}\}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\text{main}}_{j,\mathsf{f}}, \mathtt{Gacr}^{\mathsf{hash},b}) \end{aligned}$$

$$
\begin{aligned}
&+ \max_{i\in\{0,1\}} \Big[\mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{\text{main}}_{\text{sodh}}, \mathtt{Gsodh}^b) \\
&\quad + \sum_{\ell=0}^{d-1} \Big[\mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{\text{main}}_{\text{es},\ell}, \mathtt{Gxtr}^b_{es,\ell}) \\
&\qquad + \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{\text{main}}_{\text{hs},\ell}, \mathtt{Gxtr}^b_{hs,\ell}) \\
&\qquad + \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{\text{main}}_{\text{as},\ell}, \mathtt{Gxtr}^b_{as,\ell}) \\
&\quad + \sum_{n\in XPR} \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{\text{main}}_{\text{n},\ell}, \mathtt{Gxpd}^b_{n,\ell})\Big] \\
&\qquad + \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{\text{main}}_{esalt,pi}, \mathtt{Gpi}^b_{esalt}) \\
&\qquad + \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{\text{main}}_{O^*,pi}, \mathtt{Gpi}^b_{O^*})\Big] \\
&\qquad\qquad \text{Lemma E.6, E.9 and E.1} \\
&\qquad\qquad \leq \\
&\qquad \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\text{main}}_{\text{cr}}, \mathtt{Gacr}^{\mathsf{hash},b}) \\
&+ \sum_{j\in\{Z,D\},\mathsf{f}\in\{\mathsf{xtr},\mathsf{xpd}\}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\text{main}}_{j,\mathsf{f}}, \mathtt{Gacr}^{\mathsf{hash},b}) \\
&\quad + \max_{i\in\{0,1\}} \Big[\mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{\text{main}}_{\text{sodh}} \to \mathcal{R}_{\text{sodh2}}, \mathtt{Gsodh2}^b) \\
&\qquad + \sum_{alg\in\mathcal{H}} \Big(2 \cdot \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{\text{main}}_{\text{sodh}} \to \mathcal{R}_{\text{sodh-cr}}, \mathtt{Gcr}^{alg,b}) \\
&\qquad + \sum_{\ell=0}^{d-1} \Big[s_{es,\ell,alg} \cdot \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{\text{main}}_{\text{es},\ell,alg}, \mathtt{Gpr}^{\mathsf{xtr}\text{-}alg,b}) \\
&\qquad\quad + s_{hs,\ell,alg} \cdot \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{\text{main}}_{\text{hs},\ell,alg}, \mathtt{Gpr}^{\mathsf{xtr}\text{-}alg,b}) \\
&\qquad\quad + s_{as,\ell,alg} \cdot \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{\text{main}}_{\text{as},\ell,alg}, \mathtt{Gpr}^{\mathsf{xtr}\text{-}alg,b}) \\
&\qquad + \sum_{n\in XPR} s_{n,\ell,alg} \cdot \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{\text{main}}_{\text{n},\ell,alg}, \mathtt{Gpr}^{\mathsf{xpd}\text{-}alg,b})\Big] \\
&+ 2 \cdot \Big[\Big(\sum_{\ell=0}^{d-1} s_{esalt,\ell,alg}\Big) \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\text{main}}_{esalt,pi,alg}, \mathtt{Gpr}^{\mathsf{xpd}\text{-}alg,b}) \\
&\qquad\qquad + \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\text{main}}_{esalt,pi\text{-}cr,alg}, \mathtt{Gcr}^{\mathsf{xpd}\text{-}alg,b})\Big] \\
&+ \sum_{n\in SO^*\cap XPR} 2 \cdot \Big[\Big(\sum_{\ell=0}^{d-1} s_{n,\ell,alg}\Big) \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\text{main}}_{n,pi,alg}, \mathtt{Gpr}^{\mathsf{xpd}\text{-}alg,b}) \\
&\qquad\qquad + \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\text{main}}_{n,pi\text{-}cr,alg}, \mathtt{Gcr}^{\mathsf{xpd}\text{-}alg,b})\Big] \\
&+ 2^{-\mathsf{len}(alg)} \cdot \Big[(c+1)\cdot t^2_{esalt,alg} + (2c+6)\cdot t^2_{es,alg} + \sum_{n\in PO*} ((2c+6)\cdot t^2_{n,alg})\Big]\Big)\Big]
\end{aligned}
$$

This concludes the proof of Theorem 4.5. It remains to show Theorem C.1, Lemma E.1, Lemma E.6 and Lemma E.9.

C Key Schedule Theorem based on Modular Assumptions

Theorem C.1. *Let ks be a TLS-like key schedule with representative set XPR and separation points S. Let $d \in \mathbb{N}$. There is an efficient simulator $\mathcal{S}$ such that for all adversaries $\mathcal{A}$ which make queries for at most d resumption levels,*

$$
\begin{aligned}
&\mathsf{Adv}(\mathcal{A}, \mathtt{Gks}^0, \mathtt{Gks}^1(\mathcal{S})) \\
&\leq \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{main}_{cr}, \mathtt{Gacr}^{\mathsf{hash},b}) \\
&+ \sum_{j\in\{Z,D\},\mathsf{f}\in\{\mathsf{xtr},\mathsf{xpd}\}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{main}_{j,\mathsf{f}}, \mathtt{Gacr}^{\mathsf{hash},b}) \\
&\quad + \max_{i\in\{0,1\}} \big[\mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{sodh}, \mathtt{Gsodh}^b) \\
&\qquad \sum_{\ell=0}^{d-1} \big(\mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{es,\ell}, \mathtt{Gxtr}^b_{es,\ell}) \\
&\qquad\quad + \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{hs,\ell}, \mathtt{Gxtr}^b_{hs,\ell}) \\
&\qquad\quad + \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{as,\ell}, \mathtt{Gxtr}^b_{as,\ell}) \\
&\qquad + \sum_{n\in XPR} \big(\mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{n,\ell}, \mathtt{Gxpd}^b_{n,\ell})\big)\big) \\
&\qquad\quad + \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{esalt,pi}, \mathtt{Gpi}^b_{esalt}) \\
&\qquad\quad + \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{main}_{O^*,pi}, \mathtt{Gpi}^b_{O^*})\big],
\end{aligned}
$$

where $\mathcal{A}_i$ defined in Appendix D.1, modifies $\mathcal{A}$ without significantly changing its complexity: it behaves as $\mathcal{A}$ except that it returns i on some aborts; where $\mathcal{R}^{main}_* := \mathcal{R}^{ch\text{-}map} \to \mathcal{R}_*$ when replacing $*$ by cr, (Z,f), (D,f), sodh, es, hs, as, n, O^*, pi or $esalt, pi$, the simulator $\mathcal{S}$ is marked in grey in Fig. 26b, $\mathcal{R}_{sodh}$ is defined in Fig. 32a, $\mathcal{R}_{es,\ell}$ is defined in Fig. 34a, $\mathcal{R}_{hs,\ell}$ and $\mathcal{R}_{as,\ell}$ are defined analogously, and $\mathcal{R}_{n,\ell}$ for $n \in XPR$ and $0 \leq \ell \leq d$ is defined in Fig. 34b, $\mathcal{R}_{esalt,pi}$ is defined in Fig. 32c and $\mathcal{R}_{O^*,pi}$ is defined in Fig. 32d.

The proof of Theorem C.1 is outlined in Fig. 25a, 25b, 25c, 26a and 26b, each of which describes a game in a sequence of 4 game-hops which constitute the proof of Theorem 4.5. We first discuss a proof overview before turning to the proof. First, recall that the key schedule security model stores keys in a redundant fashion (a) due to possible equal values of a dishonest resumption psk ($\mathsf{level}(h) > 0$) and an adversarially registered application psk ($\mathsf{level}(h) = 0$) and (b) due to the equal values of the DH keys corresponding to (X^a, Y) and (X, Y^a).

Lemma C.2 introduces a `Map` package (see Fig. 25b for the game with the `Map` package and the left column of Fig. 29 for the code of `Map`) to remove the redundantly stored keys—note that the $\mathtt{Log}^{A1}_{psk}$ and the $\mathtt{Log}^{Z\infty}_{dh}$ package now use the $map = 1$ and the $map = \infty$ code of `Log` (see Fig. 22 for the code). As a result, any adversary playing against $\mathtt{Gcore}^0$ (defined in Fig. 25b) cannot create

(this particular) redundancy anymore since the $\texttt{Key}_{psk,\ell}$ and $\texttt{DHKey}_{dh}$ packages do not store key when the mapping code is triggered.

Lemma C.3 then reduces the indistinguishability of $\texttt{Gks}^{\texttt{0Map}}$ (Fig. 25b) and $\texttt{Gks}^{\texttt{1Map}}$ (Fig. 25c) to the indistinguishability of $\texttt{Gcore}^0$ and $\texttt{Gcore}^1(\mathcal{S}^{\text{core}})$ using reduction $\texttt{R}^{\text{core}}$. The indistinguishability of $\texttt{Gcore}^0$ and $\texttt{Gcore}^1(\mathcal{S}^{\text{core}})$ will be established in Theorem D.1 in Appendix D and contains the main technical argument of this article.

In Lemma C.4, we inline the $\texttt{Xpd}_{n,0..d}$ code into `Map` for $n \in O^*$ and call the result `Map-Xpd` (see Fig. 25c and Fig. 26a for the two games and the middle column of Fig. 29 for the code of `Map-Xpd`). The proof is a simple inlining argument.

Finally, Lemma C.5 establishes the (perfect) indistinguishability of $\texttt{Gks}^{\texttt{Map-Xpd}}$ and $\texttt{Gks}^1(\mathcal{S})$. The proof of Lemma C.5, essentially, removes or rather *inverts* the mapping on the output keys in order to recover the ideal functionality. Inverting the handle mapping, however, requires that it is *injective*. Conceptually, it is also clear that injectivity of the handle mapping needs to play a role in the proof: We prove uniqueness of output keys which means that equal keys imply equal handles. The injectivity proof ensures that the mapping did not introduce additional collisions and that the proof of Theorem D.1 indeed suffices to establish the uniqueness of output keys in $\texttt{Gks}^1(\mathcal{S})$. In summary, Lemma C.3 is the core argument, that Lemma C.2 is proven via a mechanical invariant proof, Lemma C.5 is proven via a conceptually interesting invariant proof and that Lemma C.4 is a straightforward inlining argument.

We now state each of the 4 lemmas, then show that Theorem 4.5 follows from them and Theorem D.1 and then prove each lemma in turn.

Lemma C.2 (Map-Intro). *For all adversaries $\mathcal{A}$ which make queries for at most d resumption levels,*

$$\Pr\left[1 = \mathcal{A} \to \texttt{Gks}^0\right] = \Pr\left[1 = \mathcal{A} \to \texttt{Gks}^{0Map}\right].$$

In particular $\texttt{Gks}^0 \overset{func}{\equiv} \texttt{Gks}^{0Map}$.

Lemma C.3 (Main). *For all PPT adversaries $\mathcal{A}$ which make queries for at most d resumption levels,*

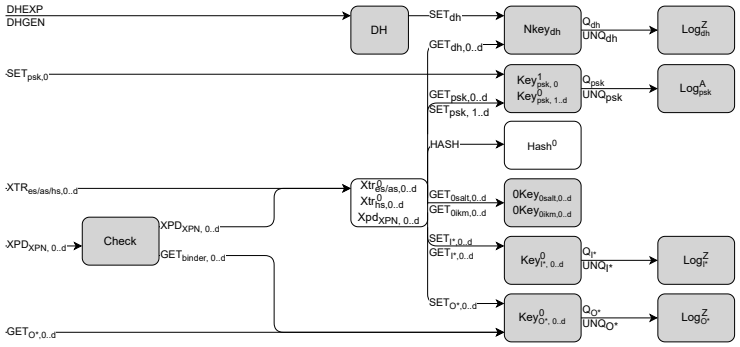
$$\begin{aligned}&\mathsf{Adv}(\mathcal{A}, \texttt{Gks}^{0Map}, \texttt{Gks}^{1Map})\\ =&\mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{ch\text{-}map}, \texttt{Gcore}^0, \texttt{Gcore}^1(\mathcal{S}^{core})),\end{aligned}$$

where $\mathcal{R}^{ch\text{-}map}$ and $\mathcal{S}^{core}$ are marked in grey in Fig. 25b.

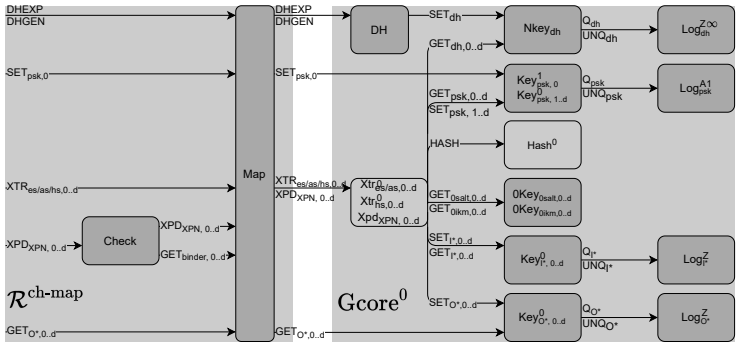
Lemma C.4 (Xpd-Inlining). *For all PPT adversaries $\mathcal{A}$ which make queries for at most d resumption levels,*

$$\Pr\left[1 = \mathcal{A} \to \texttt{Gks}^{1Map}\right] = \Pr\left[1 = \mathcal{A} \to \texttt{Gks}^{Mapxpd}\right].$$

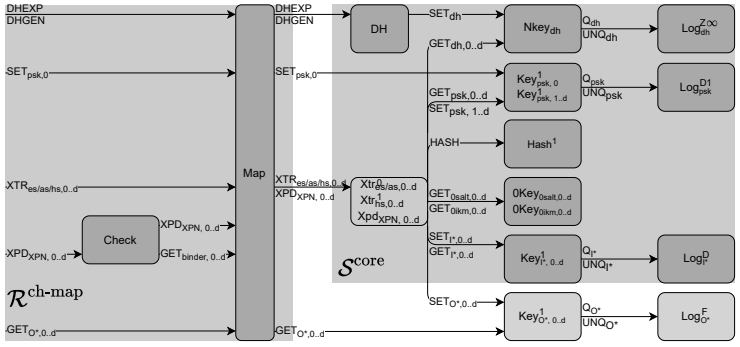
In particular $\texttt{Gks}^{1Map} \overset{code}{\equiv} \texttt{Gks}^{Mapxpd}$.



(a) Game $\texttt{Gks}^0$.

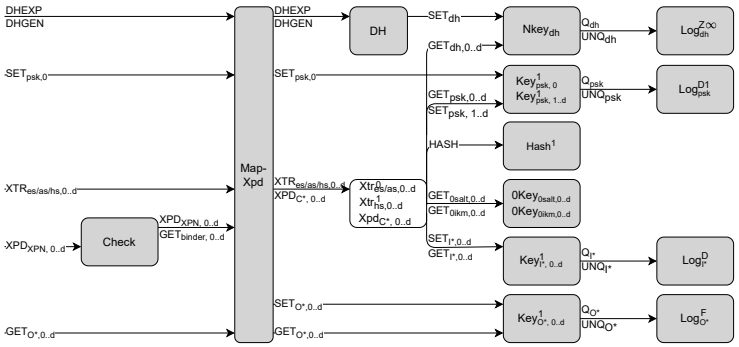


(b) Game $\texttt{Gks}^{0\text{Map}} \overset{\text{code}}{\equiv} \mathcal{R}^{\text{ch-map}} \to \texttt{Gcore}^0$.

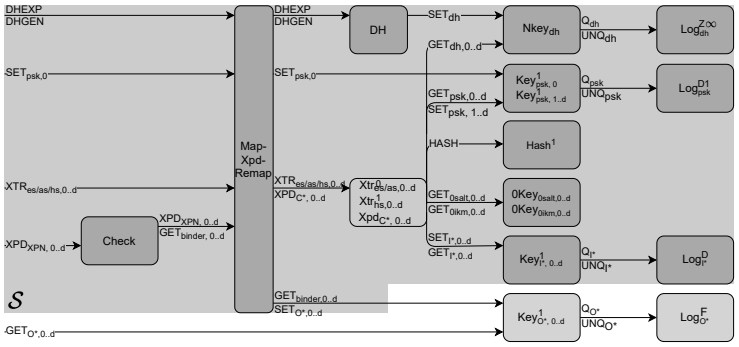


(c) Game $\texttt{Gks}^{1\text{Map}} \overset{\text{code}}{\equiv} \mathcal{R}^{\text{ch-map}} \to \texttt{Gcore}^1(\mathcal{S}^{\text{core}})$.

Fig. 25: Overview over Proof of Theorem 4.5



(a) Game $\texttt{Gks}^{\text{Mapxpd}}$.



(b) Game $\texttt{Gks}^{1}$.

Fig. 26: Overview over proof of Theorem 4.5

Lemma C.5 (Map-Outro). *For all PPT adversaries $\mathcal{A}$ which make queries for at most d resumption levels,*

$$\Pr\left[1 = \mathcal{A} \to \mathtt{Gks}^{Mapxpd}\right] = \Pr\left[1 = \mathcal{A} \to \mathtt{Gks}^{1}(\mathcal{S})\right].$$

In particular, $\mathtt{Gks}^{Mapxpd} \overset{func}{\equiv} \mathtt{Gks}^{1}(\mathcal{S})$.

Theorem 4.5 directly follows from Lemma C.2-Lemma C.5 and Theorem D.1:

$$\begin{aligned}
&\mathsf{Adv}(\mathcal{A}, \mathtt{Gks}^{0}, \mathtt{Gks}^{1}(\mathcal{S}))\\
\overset{\text{Lm. } C.2}{=}\ &\mathsf{Adv}(\mathcal{A}, \mathtt{Gks}^{0\mathrm{Map}}, \mathtt{Gks}^{1}(\mathcal{S}))\\
\overset{\text{Lm. } C.5}{=}\ &\mathsf{Adv}(\mathcal{A}, \mathtt{Gks}^{0\mathrm{Map}}, \mathtt{Gks}^{\mathrm{Mapxpd}})\\
\overset{\text{Lm. } C.4}{=}\ &\mathsf{Adv}(\mathcal{A}, \mathtt{Gks}^{0\mathrm{Map}}, \mathtt{Gks}^{1\mathrm{Map}})\\
\overset{\text{Lm. } C.3}{=}\ &\mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\text{ch-map}}, \mathtt{Gks}^{0\mathrm{core}}, \mathtt{Gks}^{1\mathrm{core}}(\mathcal{S}^{\mathrm{core}})),\\
\overset{\text{Th. } D.1}{\leq}\ &\mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\mathrm{main}}_{\mathrm{cr}}, \mathtt{Gacr}^{\mathsf{hash},b})\\
&+ \sum_{j\in\{Z,D\},\mathsf{f}\in\{\mathsf{xtr},\mathsf{xpd}\}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\mathrm{main}}_{j,\mathsf{f}}, \mathtt{Gacr}^{\mathsf{hash},b})\\
&+ \max_{i\in\{0,1\}} \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{\mathrm{main}}_{\mathrm{sodh}}, \mathtt{Gsodh}^{b})\\
&+ \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{\mathrm{main}}_{esalt,pi}, \mathtt{Gpi}^{b}_{esalt}) + \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{\mathrm{main}}_{O^*,pi}, \mathtt{Gpi}^{b}_{O^*}) +\\
&\sum_{\ell=0}^{d-1} \big(\mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{\mathrm{main}}_{\mathrm{es},\ell}, \mathtt{Gxtr}^{b}_{es,\ell}) + \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{\mathrm{main}}_{\mathrm{hs},\ell}, \mathtt{Gxtr}^{b}_{hs,\ell}) +\\
&\qquad \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{\mathrm{main}}_{\mathrm{as}}, \mathtt{Gxtr}^{b}_{as,\ell}) + \sum_{n\in XPR} \big(\mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}^{\mathrm{main}}_{\mathrm{n},\ell}, \mathtt{Gxpd}^{b}_{n,\ell})\big)\big),
\end{aligned}$$

where XPR is the representative set required by the theorem, $\mathcal{R}^{\mathrm{main}}_{*} := \mathcal{R}^{\text{ch-map}} \to \mathcal{R}_{*}$ when replacing $*$ by cr, (Z, f), (D, f) sodh, es, hs, as, n, O^*, pi or $esalt, pi$.

C.1 Proof of Lemma C.2 (Map-Intro)

In the following proof, we rely on an *index* function $\mathsf{idx}(h)$ that is equal to $\mathsf{name}(h)$ if $\mathsf{name}(h) \in \{0salt, dh, 0ikm\}$ and $\mathsf{name}(h), \mathsf{level}(h)$, else.

Invariant

(1) exponent table consistency: $E^{\text{left}} = E^{\text{right}}$

$\forall h$, let $i := \mathsf{idx}(h)$, $n := \mathsf{name}(h)$:

(2a) K and Log: for $(K_i, Log_n) = (K_i^{\text{left}}, Log_n^{\text{left}})$ and $(K_i, Log_n) = (K_i^{\text{right}}, Log_n^{\text{right}})$

(i) $\quad Log_n[h] = \bot \Rightarrow K_i[h] = \bot$

(ii) $\quad Log_n[h] = (h', *, *)$ with $h' \neq h \Rightarrow K_i[h] = \bot$

(iii) $\quad Log_n[h] = (h', hon, k)$ with $h' \neq h \Rightarrow Log_n[h'] = (h', hon, k)$

$$
\begin{array}{ll}
(iv) & Log_n[h] = (h, hon, k) \Leftrightarrow K_i[h] = (k, hon) \neq \bot \\
(v) & Log_n^{\text{left}}[h] \neq \bot \Rightarrow Log_n^{\text{left}}[h] = (h, *, *) \\
(vi) & Log_n^{\text{right}}[h], n \notin \{psk, dh\} \neq \bot \Rightarrow Log_n^{\text{right}}[h] = (h, *, *) \\
(vii) & \forall n' : Log_{n'}[h] \neq \bot \Rightarrow \mathsf{name}(h) = n' \\
(viii) & Log[h] = (h', *, k) \Rightarrow |k| = \mathsf{len}(h) = \mathsf{len}(h') \wedge \mathsf{alg}(h) = \mathsf{alg}(h') \\
(ix) & n \in \{0ikm, 0salt, dh\} \Rightarrow (K_n[h] \neq \bot \Rightarrow \mathsf{level}(h) = \bot) \\
(x) & \forall \ell : n \notin \{0ikm, 0salt, dh\} \Rightarrow (K_{n,\ell}[h] \neq \bot \Rightarrow \mathsf{level}(h) = \ell)
\end{array}
$$

(2b) Available keys (mapped-unmapped):

$$M_i^{\text{right}}[h] = h' \neq \bot \Rightarrow Log_n^{\text{right}}[h'] = (h', *, *)$$

(2c) Available keys (right-left):

$$M_i^{\text{right}}[h] = h' \neq \bot \iff Log_n^{\text{left}}[h] = (h, *, *)$$

(2d)

$$
\begin{array}{l}
M_i^{\text{right}}[h] \neq \bot \Rightarrow \\
\quad \mathsf{xtr}\langle n, h_1, h_2\rangle = h \wedge \mathsf{idx}(h_1) = i_1 \wedge \mathsf{idx}(h_2) = i_2 \Rightarrow \\
\quad\quad M_{i_1}^{\text{right}}[h_1] \neq \bot \wedge M_{i_2}^{\text{right}}[h_2] \neq \bot \wedge \\
\quad\quad M_i[h] = \mathsf{xtr}\langle n, M_{i_1}^{\text{right}}[h_1], M_{i_2}^{\text{right}}[h_2]\rangle \\
\quad \mathsf{xpd}\langle n, label, h_1, args\rangle = h \wedge \mathsf{idx}(h_1) = i_1 \wedge n \neq psk \Rightarrow \\
\quad\quad M_{i_1}^{\text{right}}[h_1] \neq \bot \wedge \\
\quad\quad M_i^{\text{right}}[h] = \mathsf{xpd}\langle n, label, M_{i_1}^{\text{right}}[h_1], args\rangle \\
\quad \mathsf{xpd}\langle psk, label, h_1, args\rangle = h \wedge \mathsf{idx}(h_1) = i_1 \Rightarrow \\
\quad\quad M_{i_1}^{\text{right}}[h_1] \neq \bot \wedge \\
\quad\quad\quad Log_{psk}[\mathsf{xpd}\langle psk, label, M_{i_1}^{\text{right}}[h_1], args\rangle] = (M_i^{\text{right}}[h], *, *)
\end{array}
$$

(2e) J-Map:

$$
\forall k : \left(\begin{array}{l}
\exists h \neq h' \wedge \mathsf{level}(h) = 0 \wedge \mathsf{level}((h') \neq 0 \wedge \\
\quad Log_{psk}^{\text{right}}[h] = (\_, 0, k) \\
\quad \wedge \ Log_{psk}^{\text{right}}[h'] = (\_, 0, k))
\end{array}\right\} \Rightarrow J_{psk}[k] = 1
$$

(3) Mapping keeps name and algs: $M_i^{\text{right}}[h] \neq \bot \Rightarrow$
(a) $\mathsf{name}(M_i^{\text{right}}[h]) = \mathsf{name}(h) \wedge$
(b) $\mathsf{alg}(M_i^{\text{right}}[h]) = \mathsf{alg}(h)$.

(4) Children derive their value from their parent(s)
For $Log_n = Log_n^{\text{left}}$ and $Log_n = Log_n^{\text{right}}$

$$
\begin{aligned}
&Log_n[h] \neq \bot \Rightarrow \\
&\quad \mathsf{xtr}\langle n, h_1, h_2\rangle = h \wedge \mathsf{name}(h_1) = n_1 \wedge \mathsf{name}(h_2) = n_2 \Rightarrow \\
&\qquad Log_{n_1}[h_1] = (h_1, hon_1, k_1) \neq \bot \wedge \\
&\qquad Log_{n_2}[h_2] = (h_2, hon_2, k_2) \neq \bot \wedge \\
&\qquad k = \mathsf{xtr}(k_1, k_2) \wedge hon = (hon_1 \vee hon_2) \wedge \\
&\qquad Log_n[h] = (*, hon, k) \\
&\quad \mathsf{xpd}\langle n, label, h_1, args\rangle = h \wedge \mathsf{name}(h_1) = n_1 \Rightarrow \\
&\qquad Log_{n_1}[h_1] = (h_1, hon_1, k_1) \neq \bot \wedge \\
&\qquad k = \mathsf{xpd}(k_1, label, args) \wedge \\
&\qquad Log_n[h] = (*, hon_1, k)
\end{aligned}
$$

(5) Consistent logs for input keys:

$$
\begin{aligned}
&n = i = dh \vee (i = (psk, 0) \wedge n = psk) \Rightarrow \\
&\quad (Log_n^{\text{left}}[h] \neq \bot) \Leftrightarrow (M_i^{\text{right}}[h] \neq \bot) \Leftrightarrow (Log_n^{\text{right}}[h] \neq \bot) \\
&\quad Log_n^{\text{left}}[h] \neq \bot \Rightarrow \\
&\qquad Log_n^{\text{left}}[h] = (h, hon, k) \wedge \\
&\qquad Log_n^{\text{right}}[h] = (M_i^{\text{right}}[h], hon', k') \wedge \\
&\qquad (hon, k) = (hon', k')
\end{aligned}
$$

(6) Identical keys and honesty:
$K_i^{\text{left}}[h] = K_{i'}^{\text{right}}[M_i^{\text{right}}[h]]$

Property (1) Exponent table: Only DHGEN writes to the exponent table and the code is identical in both games, since the Map package only forwards the call.

(2a) The proof of (2a) is a local argument over the Key and Log packages. Namely, assume that (2a) holds before a $\mathsf{SET}_{n,\ell'}$ call. We want to show that (2a) still holds. Firstly, if a write operation on $\mathtt{K}_{n,\ell}$ is made at position h, then $\mathsf{UNQ}_n(h, *, *)$ did not abort and returns $h' = h$ and as a consequence, $\mathtt{Log}_n[h]$ was written to with some value $(h, *, *)$ and thus, (2a) (i) + (ii) and the $\Leftarrow$ direction of (iv) are satisfied. Analogously, when $\mathtt{Log}_n[h] = (h, hon, k)$ is written, then $K_{n,\ell}[h]$ is written to with (k, hon) and thus also the $\Rightarrow$ direction of (iv) follows.

For (iii), observe that if UNQ_n writes (h', hon, k) into $Log_n[h]$ for some $h' \neq h$, then one of the *map* patterns was active and found an entry $Log_n[h'] = (h', hon, k)$, thus, this entry exists in $Log_n[h'] = (h', hon, k)$ and (iii) is satisfied.

For (v) and (vi) observe that the left $\mathtt{Log}_n$ packages do not have a *map* pattern. Similarly the right $\mathtt{Log}_n$ packages for $n \notin \{dh, psk\}$ do not have a *map* pattern.

For (vii), since $\mathsf{SET}_{i'}(h, hon, k)$ for $i' = n' = dh$ or $i' = (n', \ell)$ asserts that $\mathsf{name}(h) = n'$, all handles h which appear in $\mathsf{UNQ}_n(h, hon, k)$ indeed have name n and thus $n)\mathsf{name}(h)$.

For (viii), consider a $\mathsf{SET}_{n,\ell}(h, hon, k)$ query. Since $\mathsf{SET}_{n,\ell}$ checks that $|k| = \mathsf{len}(h)$, if a write operation on Log_n is performed (i.e., $\mathsf{Q}_n(h)$ returns $\perp$), then for $Log_n[h] = (h', hon, k)$, we have that $|k| = \mathsf{len}(h)$ as required. If $h = h'$, then $\mathsf{alg}(h) = \mathsf{alg}(h')$. If $h \neq h'$, then observe that 2a (iii) implies that $Log_n[h'] = (h', hon, k)$ and thus $\mathsf{len}(h') = |k|$. Since mapping from algorithms to length is injective, this also implies that $\mathsf{alg}(h') = \mathsf{alg}(h)$. Similarly for (x), $\mathsf{SET}_{n,\ell}(h, hon, k)$ asserts $\mathsf{level}(h) = \ell$.

Finally, for (ix) observe that for $n \in \{\mathit{0salt}, \mathit{0ikm}\}$, $\mathsf{SET}_n(h, hon, k)$ is never used and the only values in K are those set by the initialization. For dh observe that the $\mathsf{SET}_n(h, hon, k)$ is used only in the `DH` package which only generates handles of the form $\mathsf{dh}\langle\mathsf{sort}(X, Y)\rangle$ which are of level 0 as required. This concludes the proof of (2a).

(2b) Note that the handles on the right-hand sides of $M_{i'}$ are returned by a $\mathsf{SET}_{n,\ell}$ query (for a potentially different index $i := (n, \ell)$) on a `Key` package. Now, we claim that for all handles h' which are returned by $\mathsf{SET}_{n,\ell}$, we have that $\mathtt{Log}_n[h'] = (h', *, *)$. Firstly, when $\mathsf{Q}_n(h)$ returns a handle h', then by 2a (iii), we have that $\mathtt{Log}_n[h'] = (h', *, *)$. In turn, if $\mathsf{UNQ}_n(h, hon, k)$ returns a $h' \neq h$, then $\mathtt{Log}_n[h] = (h', hon, k)$ and by 2a (iii), $\mathtt{Log}_n[h'] = (h', *, *)$. In turn, if $\mathsf{UNQ}_n(h, hon, k)$ returns a h, then $\mathtt{Log}_n[h] = (h, hon, k)$ and we also return h in the end of the $\mathsf{SET}_{n,\ell}$ call.

(2c) Code inspection yields that if $M_{n,\ell}[h]$ is written to on the right, then on the left, we make a $\mathsf{SET}_{n,\ell}(h, *, *)$ query, and conversely. After the $\mathsf{SET}_{n,\ell}(h, *, *)$ query, $\mathtt{Log}_n[h]$ is defined and by 2a (v), it is equal to $\mathtt{Log}_n[h] = (h, *, *)$.

(2d)

$\mathsf{XTR}_{n,\ell}$ First observe that in the $\mathsf{XTR}_{n,\ell}$ oracles, the code of `Map` asserts that for $i_1, i_2 \leftarrow \mathsf{PrntIdx}(n, \ell)$ both $M_{i_1}^{\text{right}}[h_1] \neq \perp$ and $M_{i_2}^{\text{right}}[h_2] \neq \perp$. Moreover, the $\mathsf{XTR}_{n,\ell}$ oracle of `Map` constructs as handle $h = \mathsf{xtr}\langle n, h_1, h_2\rangle$ and calls its own $\mathsf{XTR}_{n,\ell'}$ with handles $(M_{i_1}^{\text{right}}[h_1], M_{i_2}^{\text{right}}[h_2])$ and level computed as $\ell' \xleftarrow{\text{choose}} \mathsf{level}(M_{i_1}^{\text{right}}[h_1]), \mathsf{level}(M_{i_2}^{\text{right}}[h_2])$. Oracle $\mathsf{XTR}_{n,\ell'}$ returns handle $h = \mathsf{xtr}\langle n, M_{i_1}^{\text{right}}[h_1], M_{i_2}^{\text{right}}[h_2]\rangle$, since it returns the handle it constructs: The $\mathtt{Key}_{n,\ell'}^{\text{right}}$ packages to which $\mathsf{XTR}_{n,\ell'}$ writes does not change the handle but returns it unmodified (condition (2a) (vi)).

$\mathsf{XPD}_{n,\ell}$ If $n \neq psk$, the analysis is analogous to $\mathsf{XTR}_{n,\ell}$. For $n = psk$, the only difference is that the $\mathsf{SET}_{psk,\ell'}$ query of the $\mathsf{XPD}_{psk,\ell'}$ oracle goes to the $\mathtt{Key}_{psk,\ell'}$ package which has an active mapping in the $\mathtt{Log}_{psk}^{\text{right}}$ package. Since the $\mathtt{Key}_{psk}$ package, on query $\mathsf{SET}_{psk,\ell'}(h, hon, k)$, returns a handle h^* such that $\mathtt{Log}_{psk}^{\text{right}}[h] = (h^*, *, *)$ and since `Map` writes $M_{n,\ell}^{\text{right}}[h] \leftarrow h^*$, we have that indeed $\mathtt{Log}_n^{\text{right}}[h] = (M_{n,\ell}^{\text{right}}[h], *, *)$ after the call as required.

(2e) First observe that J_{psk} is monotonous, i.e., if $J_{psk}[k] = 1$, it is never set to a different value than 1. The interesting case is when before a $\mathsf{SET}_{psk,\ell}(h, hon, k)$ call, $J_{psk}[k] = \bot$ and there exists a handle $h' \neq h$ such that $\mathsf{level}(h) = 0 \wedge \mathsf{level}((h') \neq 0$ or $\mathsf{level}(h') = 0 \wedge \mathsf{level}(h) \neq 0$ and $Log^{\text{right}}_{psk}[h'] = (\_, 0, k)$. In this case, the *map* code for $map = 1$ will return 1 and thus, $J_{psk}[k] \leftarrow 1$ is written.

(3) Assuming that (3a) holds so far, consider a new call to $\mathsf{XPD}_{n,\ell}$ or $\mathsf{XTR}_{n,\ell}$ and let $(i_1, i_2) \leftarrow \mathsf{PrntIdx}(n, \ell)$. Since the handles are constructed based on previous handles, and the name function is defined based on the handle construction (2d), $\mathsf{xpd}\langle n, M^{\text{right}}_{i_1}[h_1], args\rangle$ and $\mathsf{xpd}\langle n, h_1, args\rangle$ have the same name if $\mathsf{name}(h_1) = \mathsf{name}(M^{\text{right}}_{i_1}[h_1])$ (else, either the name of $\mathsf{xpd}\langle n, M^{\text{right}}_{i_1}[h_1], args\rangle$ or the name of $\mathsf{xpd}\langle n, h_1, args\rangle$ would not be well-defined, since the name of h_1 needs to be parent of n) which holds by induction hypothesis. The only exception is $\mathsf{XPD}_{psk,1..d}$ and $\mathsf{SET}_{psk,0}$ in the case that a mapping in the $\mathtt{Log}_{psk}$ package occurs. By induction hypothesis (condition 2a (v) and (vii)), all handles already stored in $\mathtt{Log}_{psk}$ have name equal to psk, and since $\mathtt{Log}^{\text{right}}_{psk}$ returns a handle which is already stored in $\mathtt{Log}^{\text{right}}_{psk}$, this handle also has name psk.

Analogously to the XPD case, the induction hypothesis implies that $\mathsf{name}(h_1) = \mathsf{name}(M^{\text{right}}_{i_1}[h_1])$ and $\mathsf{name}(h_2) = \mathsf{name}(M[h_2])$ and thus, $\mathsf{xtr}\langle n, h_1, h_2\rangle$ and $\mathsf{xtr}\langle n, M^{\text{right}}_{i_1}[h_1], M^{\text{right}}_{i_1}[h_2]\rangle$ have the same name.

Assuming (3b) holds so far, we proceed by induction over the oracle calls. Let $i := \mathsf{idx}(h)$ and $(i_1, \_) \leftarrow \mathsf{PrntIdx}(i)$. By induction hypothesis, $alg = \mathsf{alg}(h_1) = \mathsf{alg}(M^{\text{right}}_{i_1}[h_1])$. Moreover, as handles inherit their algorithm from the first psk, we also have

$$\mathsf{alg}(\mathsf{xpd}\langle n, label, h_1, args\rangle) = \mathsf{alg}(h_1)$$
$$\mathsf{alg}(h_1) = \mathsf{alg}(M^{\text{right}}_{i_1}[h_1])$$

Putting these together with 2d, for $n \neq psk$, we obtain:

$$\begin{aligned}
&\mathsf{alg}(\mathsf{xpd}\langle n, label, h_1, args\rangle) \\
=&\mathsf{alg}(h_1) && \text{(parent-child)} \\
=&\mathsf{alg}(M^{\text{right}}_{i_1}[h_1]) && \text{(induction hypothesis)} \\
=&\mathsf{alg}(\mathsf{xpd}\langle n, label, M^{\text{right}}_{i_1}[h_1], args\rangle) && \text{(child-parent)} \\
=&\mathsf{alg}(M^{\text{right}}_{i}[\mathsf{xpd}\langle n, label, h_1, args\rangle]) && (2d)
\end{aligned}$$

The analysis for xtr is analogous, except that we need to apply the induction hypothesis on both parent handles and rely on algorithm consistency of the parent handles, i.e., either both have the same algorithm or only one has an algorithm since the `Xtr` package code asserts $\mathsf{alg}(h_1) = \mathsf{alg}(h_2)$ if both are defined.

We now turn to the case that $n = psk$. If $M^{\text{right}}_{psk,\ell}[\mathsf{xpd}\langle n, label, h_1, args\rangle] = h' = \mathsf{xpd}\langle n, label, M^{\text{right}}_{i_1}[h_1], args\rangle$, the analysis is analogous. Else if $h' \neq \mathsf{xpd}\langle n, label, M^{\text{right}}_{i_1}[h_1], args\rangle$, then due to 2d, $Log^{\text{right}}_{psk}[h] = (h', *, *)$ and by 2a (viii),

$\mathsf{alg}(h) = \mathsf{alg}(h')$ as required. Lastly, let us consider $\mathsf{SET}_{psk,0}(h, hon, k)$ and here, we consider the case that a write operation on Log_{psk} is performed, i.e. $Log^{\text{right}}_{psk}[h] = (h', hon, k)$ and note that $M^{\text{right}}_{psk,0}[h]$ then contains h'. Due to 2a (viii), $\mathsf{alg}(h) = \mathsf{alg}(h') = \mathsf{alg}(M_{psk,0}[h])$ as required.

(4) Assume (4) holds as induction hypothesis. We only need to consider the case that a write operation on $\mathtt{Log}_n$ is performed. Note that we do not need argue about $\mathtt{Map}$. Instead, we only argue over $\mathtt{Xpd}_{n,\ell}$ and $\mathtt{Xtr}_{n,\ell}$ and the $\mathtt{Key}_{n,\ell}$ and $\mathtt{Log}_n$ packages they call. Let $(n_1, n_2) \leftarrow \mathsf{PrntN}(n)$. In an $\mathsf{XPD}_{n,\ell}(h_1, label, args)$ call, $\mathsf{GET}_{n_1}(h_1)$ returns a handle only if $Log_{n_1}[h_1] = (h_1, hon_1, k_1)$ (condition 2a iv). Thus, when $\mathsf{XPD}_{n,\ell}(h_1, label, args)$ makes a $\mathsf{SET}_{n,\ell}(h, hon, k)$ query (which induces a write operation on the Log_n), then $Log_{n_1}[h_1] = (h_1, hon_1, k_1) \neq \bot$, as required. Moreover, $\mathsf{XPD}_{n,\ell}(h_1, label, args)$ computes key k as $\mathsf{xpd}(k_1, (label, args))$, computes handle h as $\mathsf{xpd}\langle n, label, h_1, args\rangle$ and sets honesty $hon := hon_1$. Thus, $Log_n[h]$ stores a triple $(*, hon, k)$ as required. The argument for $\mathsf{XTR}_{n,\ell}$ is analogous.

(5) The SET_{dh} queries are identical on both sides, as $\mathtt{Map}$ forwards all DHEXP queries. Therefore, the same UNQ queries are made to $\mathtt{Log}^{\text{right}}_{dh}$ and $\mathtt{Log}^{\text{left}}_{dh}$ and thus,

$$\mathtt{Log}^{\text{left}}_{dh}[h] \neq \bot \Leftrightarrow \mathtt{Log}^{\text{right}}_{dh}.$$

Since $\mathsf{SET}_{0,psk}(h, hon, k)$ is forwarded by $\mathtt{Map}$ and these are the only queries to $\mathtt{Log}_{psk}$ which use a handle h with $\mathsf{level}(h) = 0$ (since $\mathtt{Key}_{psk,\ell}$ asserts that $\mathsf{level}(h) = \ell$) we have that for handles h with $\mathsf{level}(h) = 0$, the same queries are made to $\mathtt{Log}^{\text{left}}_{psk}$ and $\mathtt{Log}^{\text{right}}_{psk}$ and thus,

$$\mathsf{level}(h) = 0 \Rightarrow \mathtt{Log}^{\text{left}}_{psk}[h] \neq \bot \Leftrightarrow \mathtt{Log}^{\text{right}}_{psk}.$$

In case that $\mathtt{Log}^{\text{left}}_{n}[h] \neq \bot$, by 2a (v), $\mathtt{Log}^{\text{left}}_{n}[h] = (h, *, *)$. Moreover, since the queries are the same on both sides, the hon and k values are identical, i.e., if $\mathtt{Log}^{\text{left}}_{n}[h] = (h, hon, k)$ and $\mathtt{Log}^{\text{right}}_{n}[h] = (h', hon', k')$, then $(hon, k) = (hon', k')$. Moreover, since $\mathtt{Log}^{\text{right}}_{n}[h]$ returns h' and $\mathtt{Key}_{psk,\ell}$ (resp. $\mathtt{NKey}_{dh}$) returns h' to $\mathtt{Map}$, we also have that $M^{\text{right}}_{i}[h] \leftarrow h'$ and thus, $h' = M^{\text{right}}_{i}[h]$.

Property (6)

$\mathsf{SET}_{psk,0}(h, hon, k)$**:** $\mathsf{SET}_{psk,0}(h, hon, k)$ changes at most level 0 entries in Log^{left}_{psk}, Log^{right}_{psk} and $M^{\text{right}}_{psk,0}$ due to the level assert in $\mathsf{SET}_{n,\ell}$. By (5), these three are equivalent. If no changes are made, then condition (6) holds by induction hypothesis. If changes are made, then $M^{\text{right}}_{psk,0}[h]$, $Log^{\text{left}}_{psk}[h]$ and $Log^{\text{right}}_{psk}[h]$ are defined. By (5), we have that $Log^{\text{left}}_{psk}[h] = (h, k, hon)$ and thus, by (2a iv), we also have that $K^{\text{left}}_{psk}[h] = (k, hon)$. Moreover, by (5), $M^{\text{right}}_{psk,0}[h] \neq \bot$ and $Log^{\text{right}}_{psk}[h] = (M^{\text{right}}_{psk,0}[h], k', hon')$ and $(k', hon') = (k, hon)$. By 2a (iii), we have that $Log^{\text{right}}_{psk}[M^{\text{right}}_{psk,0}[h]] = (M^{\text{right}}_{psk,0}[h], k, hon)$ and thus, by 2a (iv), $K^{\text{right}}_{psk}[M^{\text{right}}_{psk,0}[h]] = (k, hon)$, as required.

DHGEN: Invariant (6) does not refer to E.

DHEXP: Oracle $\mathsf{DHEXP}(X, Y)$ changes at most entries in Log_{dh}^{left}, Log_{dh}^{right} and M_{dh}^{right}. By (5), these three are equivalent. If no changes are made, then condition (6) holds by induction hypothesis. If changes are made, then $M_{dh}^{\text{right}}[h]$, $Log_{dh}^{\text{left}}[h]$ and $Log_{dh}^{\text{right}}[h]$ are defined. By (5), we have that $Log_{dh}^{\text{left}}[h] = (h, k, hon)$ and thus, by (2a iv), we also have that $K_{dh}^{\text{left}}[h] = (k, hon)$. Moreover, by (5), $M_{dh}^{\text{right}}[h] \neq \bot$ and $Log_{dh}^{\text{right}}[h] = (M_{dh}^{\text{right}}[h], k', hon')$ and $(k', hon') = (k, hon)$. By 2a (iii), we have that $Log_{dh}^{\text{right}}[M_{dh}^{\text{right}}[h]] = (M_{dh}^{\text{right}}[h], k, hon)$ and thus, by 2a (iv), $K_{dh}^{\text{right}}[M_{dh}^{\text{right}}[h]] = (k, hon)$, as required.

$\mathsf{XTR}_{n,\ell}$: Let $h := \mathsf{xtr}\langle h_1, , h_2\rangle$ be the handle which $\mathsf{XTR}_{n,\ell}(h_1, h_2)$ returns. First observe that $M_{n,\ell}^{\text{right}}$ remains unchanged except for position $M_{n,\ell}^{\text{right}}[h]$ which might either be overwritten or be written to for the first time. We now consider each of the two cases in turn.

Case I: If $M_{n,\ell}^{\text{right}}[h]$ already contained a value h', then due to (2d), before and after the call, we have that $M_{n,\ell}^{\text{right}}[h]$ is equal to $\mathsf{xtr}\langle M[h_1], M[h_2]\rangle$ and thus $M_{n,\ell}^{\text{right}}[h]$ remains unchanged. Moreover observe that $K_{n,*}^{\text{right}}$ is never overwritten: Only $\mathsf{SET}_{n,*}^{\text{right}}$ makes a write access to $K_{n,*}[h]$. If $K_{n,*}[h] \neq \bot$, then, by (2a i), $Log_n[h] \neq \bot$ and thus, Q_n returns on a $\mathsf{SET}_{n,*}$ query before a write to $K_{n,*}$ is performed. Let $\ell' = \mathsf{level}(M_{n,\ell}^{\text{right}}[h])$, then, since 2c and 2a guarantee that $K_{n,\ell}^{\text{left}}[h]$, $M_{n,\ell}^{\text{right}}[h]$ and $K_{n,\ell'}^{\text{right}}[M_{n,\ell}^{\text{right}}[h]]$ have been defined before the current call, no state is changed and (6) follows by induction hypothesis.

Case II: If $M_{n,\ell}^{\text{right}}[h]$ did not contain a value before this call, then we rely on (2d) to see that $M_{n,\ell}^{\text{right}}[h] = \mathsf{xtr}\langle M_{i_1}^{\text{right}}[h_1], M_{i_2}^{\text{right}}[h_2]\rangle$ for $(i_1, i_2) \leftarrow \mathsf{PrntIdx}(n, \ell)$ and $(n_1, n_2) \leftarrow \mathsf{PrntN}(n)$. Since $M_{i_1}[h_1]$ and $M_{i_2}[h_2]$ were defined before the call already.

– Using (4) and (2a (v)), we know that $Log_n^{\text{left}}[h] = (h, hon^{\text{left}}, k^{\text{left}})$ with $k^{\text{left}} = xtr(K_{n_1,*}^{\text{left}}[h_1], K_{n_2,*}^{\text{left}}[h_2])$, and

– using (4) and (2a (vi)), we know that $Log_n^{\text{right}}[M_i^{\text{right}}[h]] = (M_i^{\text{right}}[h], hon^{\text{right}}, k^{\text{right}})$ with $k^{\text{right}} = xtr(K_{n_1,*}^{\text{right}}[M_{i_1}[h_1]], K_{n_2,*}^{\text{right}}[M_{i_2}[h_2]])$.

– Using (2a (iv)), we also know that $K_{n,\ell}^{\text{left}}[h] = (k^{\text{left}}, hon^{\text{left}})$ and $K_{n,*}^{\text{right}}[h] = (k^{\text{right}}, hon^{\text{right}})$.

– Using (6) as induction hypothesis, we obtain that $k_1, hon_1 = K_{n_1,\ell}^{\text{left}}[h_1] = K_{n_1,*}^{\text{right}}[M_{n_1,\ell}^{\text{right}}[h_1]]$ and $k_2, hon_2 = K_{n_2,\ell}^{\text{left}}[h_1] = K_{n_2,*}^{\text{right}}[M_{n_2,\ell}^{\text{right}}[h_1]]$.

Therefore, as required:

$$
\begin{aligned}
K_{n,\ell}^{\text{left}}[h] &= (k^{\text{left}}, hon^{\text{left}}) \\
&= (\mathsf{xtr}(K_{n_1,\ell}^{\text{left}}[h_1].k, K_{n_2,\ell}^{\text{left}}[h_2].k), K_{n_1,\ell}^{\text{left}}[h_1].hon \vee K_{n_2,\ell}^{\text{left}}[h_2].hon) \\
&= (\mathsf{xtr}(k_1, k_2,), hon_1 \vee hon_2) \\
&= (\mathsf{xtr}(K_{n_1,*}^{\text{right}}[M_{i_1}[h_1]].k, K_{n_2,*}^{\text{right}}[M_{i_2}[h_2]].k), \\
&\qquad K_{n_1,*}^{\text{right}}[M_{i_1}[h_1]].hon \vee K_{n_2,*}^{\text{right}}[M_{i_2}[h_2]].hon) = K_{n,*}^{\text{right}}[M_{n,\ell}[h]]
\end{aligned}
$$

$\mathsf{XPD}_{n,\ell}$**:** If $n \neq psk$, then the analysis is analogous to $\mathsf{XTR}_{n,\ell}$. Let us now consider $\mathsf{XPD}_{psk,\ell}$ with $h = \mathsf{xpd}\langle psk, label, h_1, args\rangle$. Now, in the analysis of $\mathsf{XPD}_{psk,\ell}$, condition (2d) yield that either $M^{\text{right}}_{psk,\ell}[h] = \mathsf{xpd}\langle psk, label, M^{\text{right}}_{rm,\ell-1}[h_1], args\rangle$ or some $h^\star \neq \mathsf{xpd}\langle psk, label, M^{\text{right}}_{rm,\ell-1}[h_1], args\rangle$. In the case that $M^{\text{right}}_{psk,\ell}[h]$ is equal to $\mathsf{xpd}\langle psk, label, M^{\text{right}}_{rm,\ell-1}[h_1], args\rangle$ (level >0), the analysis is analogous to $\mathsf{XTR}_{n,\ell}$, except that $Log^{\text{right}}_{psk}[h] = (h, *, *)$ now follows from (2d) instead of (2a) vi (since psk is one of the exceptions specified in (2a)). Let us now turn to the case that $M^{\text{right}}_{psk,\ell}[h] = h^\star \neq \mathsf{xpd}\langle psk, label, M^{\text{right}}_{rm,\ell-1}[h_1], args\rangle$ (level=0). By condition 2d $M^{\text{right}}_{rm,\ell-1}[h_1] \neq \perp$ and $K^{\text{right}}_{rm,*}[M^{\text{right}}_{rm,\ell-1}[h_1]] = (k_1, hon_1) \neq \perp$. Denote by $hon := hon_1$ and $k := \mathsf{xpd}(k_1, args)$. We obtain the following two equations:

$$Log^{\text{right}}_{psk}[\mathsf{xpd}\langle psk, label, M^{\text{right}}_{rm,\ell-1}[h_1], args\rangle] = (h^\star *, *) \quad \text{(due to 2d)} \quad (14)$$

$$Log^{\text{right}}_{psk}[\mathsf{xpd}\langle psk, label, M^{\text{right}}_{rm,\ell-1}[h_1], args\rangle] = (*, hon, k) \quad \text{(due to 4)} \quad (15)$$

$$\Rightarrow Log^{\text{right}}_{psk}[\mathsf{xpd}\langle psk, label, M^{\text{right}}_{rm,\ell-1}[h_1], args\rangle] = (h^\star, hon, k) \quad \text{(Eq.14+15)} \quad (16)$$

$$Log^{\text{right}}_{psk}[h^\star] = (h^\star, hon, k) \quad \text{(2a iii+Eq.16)} \quad (17)$$

$$\Rightarrow K^{\text{right}}_{psk,\mathsf{level}(h^\star)}[h^\star] = (k, hon) \quad \text{(2a iv+Eq.17)}$$

$$\Rightarrow K^{\text{right}}_{psk,\mathsf{level}(M^{\text{right}}_{psk}[h])}[M^{\text{right}}_{psk}[h]] = (k, hon)$$

Analogously to $\mathsf{XTR}_{n,\ell}$, we have $K^{\text{right}}_{n_1,\mathsf{level}(M^{\text{right}}_{i_1}[h_1])}[M^{\text{right}}_{i_1}[h_1]] = (k_1, hon_1) \neq \perp$ implies that $K^{\text{left}}_{n_1,*}[h_1] = (k_1, hon_1)$. Using (6) on h_1 and (4), we obtain that $K^{\text{left}}_{n,\ell}[h] = (\mathsf{xpd}(k_1, args), hon_1) = (k, hon)$, as required.

$\mathsf{GET}_{n,\ell}$: Since $\mathsf{GET}_{n,\ell}$ does not perform write operations, the invariant follows from the induction hypothesis.

Same input-output-behaviour We want to prove that if the invariant holds, then the left game and the right game give identical answers to each query. We consider each query separately in turn and first argue that the left game aborts if and only if the right game aborts. We then argue that if the game do not abort, then they return the same answer.

DHGEN There are no asserts in the code, and the oracle always returns the public share.

DHEXP The assert that $\mathsf{grp}(X) = \mathsf{grp}(Y)$ is identically evaluated in both games. The assert $E[X] \neq \perp$ is evaluated consistently in both games due to invariant property (1) which states that $E^{\text{left}} = E^{\text{right}}$. Both games return the same pair of public shares which they received as input (regardless of whether mapping occurs).

$\mathsf{SET}_{psk,0}(h, hon, k)$ `Map` passes on $\mathsf{SET}_{psk,0}(h, hon, k)$ unchanged and the four asserts in the $\mathsf{SET}_{psk,0}(h, hon, k)$ are evaluated on the same inputs and do not rely on state, i.e., **assert** $\mathsf{name}(h) = psk$, **assert** $\mathsf{level}(h) = 0$, **assert** $\mathsf{alg}(k) = \mathsf{alg}(h)$ and **assert** $\mathsf{len}(h) = |\mathsf{untag}(k^\star)|$. Thus, either both or

none of the two games abort. If they don't abort, let us consider the following cases.

Case I there are no level 0 entries for $(0, k)$ in Log^{right}_{psk}: Due to (5), there are also no level 0 entries for k in Log^{left}_{psk}. Thus, none of the two games aborts.

Case II there is a level 0 entry for $(0, k)$ in Log^{right}_{psk}: Due to (5), there is also a level 0 entry in Log^{left}_{psk}.

(a) There is no level > 0 entry for $(0, k)$ in Log^{right}_{psk}. Therefore, the $map = 1$ code of the $\mathtt{Log}^{\text{right}}_{psk}$ package does not trigger and both games abort.

(b) There is a level > 0 entry for $(0, k)$ in Log^{right}_{psk}. By (2e), we have that $J[k] = 1$. Therefore, the $map = 1$ code of the $\mathtt{Log}^{\text{right}}_{psk}$ package does not trigger and both games abort.

Moreover, if there is no abort, both games return the same handle as they received. In the left game, this is ensured by the Key package code, and in the right game, this is ensured by the Map code.

$\mathsf{XTR}_{n,\ell}/\mathsf{XPD}_{n,\ell}$ For $\mathsf{XTR}_{n,\ell}$, consider the assert **assert** $\mathsf{alg}(h_1) = \mathsf{alg}(h_2)$ which is evaluated if $\mathsf{alg}(h_1) \neq \bot$ and $\mathsf{alg}(h_2) \neq \bot$. Since 3(b) ensures that $M_{n,\ell}$ preserves algorithms (including the case that algorithms are undefined), this assert fails on the right if and only if fails on the left.

We now consider the asserts when reading keys in the $\mathsf{XTR}_{n,\ell}$ oracle of the $\mathtt{Xtr}$ package via GET queries. Where the left game reads a keys for handles h_1 and h_2, the right game reads keys for $M^{\text{right}}_{i_1}[h_1]$ and $M^{\text{right}}_{i_2}[h_2]$, where $i_1 = \mathsf{idx}(h_1)$ and $i_2 = \mathsf{idx}(h_2)$. Due to 2c (and using 2a (iv)), $M^{\text{right}}_{i_1}[h_1]$ and $M^{\text{right}}_{i_2}[h_2]$ are defined if and only if $K^{\text{left}}_{i_1}[h_1]$ and $K^{\text{left}}_{i_2}[h_2]$ are defined, and, due to 2b (and using 2a (iv)), when $M^{\text{right}}_{i_1}[h_1] \neq \bot$ and $M^{\text{right}}_{i_2}[h_2] \neq \bot$, then also $K^{\text{right}}_{\mathsf{idx}(M^{\text{right}}_{i_1}[h_1])}[M^{\text{right}}_{i_1}[h_1]] \neq \bot$ and $K^{\text{right}}_{\mathsf{idx}(M^{\text{right}}_{i_2}[h_2])}[M^{\text{right}}_{i_2}[h_2]] \neq \bot$. The read asserts for $\mathsf{XPD}_{n,\ell}$ are analogous.

We now consider asserts when writing keys in the $\mathsf{XTR}_{n,\ell}$ oracle via SET queries. Here, the analysis of asserts is almost analogous to $\mathsf{SET}_{psk,0}(h, hon, k)$, except that now, some of the values used in the asserts have previously been retrived via a GET query. Firstly, **assert** $\mathsf{name}(h) = n$ holds, since h is constructed as $\mathsf{xtr}\langle n, h_1, h_2\rangle$ and h_1 and h_2 have the correct name due to 2a (vii) (and 2a (iv)), thus, h is a valid handle with $\mathsf{name}(h) = n$. Analogously, the level of a handle is inherited from the level of one of the parent handles, and since only one of the parent handles has a level (neither 0ikm, nor 0salt, nor dh have a level due to 2a (ix)) and this level is the same as the level of the $\mathtt{Xtr}_{n,\ell}$ package (due to the code of $\mathsf{XTR}_{n,\ell}$ and 2a (x)), if $\mathsf{XTR}_{n,\ell}$ makes a $\mathsf{SET}_{n,\ell}(h, hon, k)$ query, then $\mathsf{level}(h) = \ell$ and **assert** $\mathsf{level}(h) = \ell$ is satisfied. For **assert** $\mathsf{alg}(k) = \mathsf{alg}(h)$, we observe that GET tags keys consistently with the algorithm contained in the handle. Finally, **assert** $\mathsf{len}(h) = |\mathsf{untag}(k^\star)|$ is satisfied since 2(a) (viii) (and 2a (iv)) is satisfied on a GET and xtr preserves the key length.

The asserts for $\mathsf{XPD}_{n,\ell}$ are analogous, except for $n = psk$ where the level is increased by 1 before setting the key in the XPD code on the right and on the left. Since $\mathsf{level}(\mathsf{xpd}\langle psk, label, h_1, args\rangle) = \mathsf{level}(h_1) + 1$, this increase is consistent between the handle and the $\mathsf{SET}_{psk,\ell+1}$ query.
If an $\mathsf{XTR}_{n,\ell}$ query does not abort, then $\mathsf{XTR}_{n,\ell}(h_1, h_2)$ returns $\mathsf{xtr}\langle n, h_1, h_2\rangle$ on both sides, since this handle is constructed by `Map` on the right and by `Xtr` on the left. Analogously for $\mathsf{XPD}_{n,\ell}(h, label, args)$, the call returns $\mathsf{xpd}\langle n, label, h_1, args\rangle$ on both sides whenever it does not abort.

$\mathsf{GET}_{n,\ell}$ Due to (2b, 2c and 2a (iv)), the entry $M^{\text{right}}_{n,\ell}[h]$ is defined if and only if the values $K^{\text{right}}_{n,\mathsf{level}(M^{\text{right}}_{n}[h])}[M^{\text{right}}_{n,\ell}[h]]$ and $K^{\text{left}}_{n,\ell}[h]$ are defined. It follows from (6) that the same value is returned.

C.2 Proof of Lemma C.3 (Main)

Observe that

$$\mathtt{Gks}^{0\text{Map}} \overset{\text{code}}{\equiv} \mathcal{R}^{\text{ch-map}} \to \mathtt{Gcore}^0 \tag{18}$$

$$\mathtt{Gks}^{1\text{Map}} \overset{\text{code}}{\equiv} \mathcal{R}^{\text{ch-map}} \to \mathtt{Gcore}^1(\mathcal{S}^{\text{core}}) \tag{19}$$

in Fig. 25b and Fig. 25c, respectively. Therefore, based on $\mathcal{A}$, we can define a new adversary $\mathcal{B} := \mathcal{A} \to \mathcal{R}^{\text{ch-map}}$ in the analysis:

$$\begin{aligned}
& \mathsf{Adv}(\mathcal{A}, \mathtt{Gks}^{0\text{Map}}, \mathtt{Gks}^{1\text{Map}}) \\
= \;& \left|\Pr\left[1 = \mathcal{A} \to \mathtt{Gks}^{0\text{Map}}\right] - \Pr\left[1 = \mathcal{A} \to \mathtt{Gks}^{1\text{Map}}\right]\right| \\
\overset{\text{Eq.18\&19}}{=} \;& \left|\Pr\left[1 = \mathcal{A} \to (\mathcal{R}^{\text{ch-map}} \to \mathtt{Gcore}^0)\right] - \Pr\left[1 = \mathcal{A} \to (\mathcal{R}^{\text{ch-map}} \to \mathtt{Gcore}^1(\mathcal{S}^{\text{core}}))\right]\right| \\
= \;& \left|\Pr\left[1 = (\mathcal{A} \to \mathcal{R}^{\text{ch-map}}) \to \mathtt{Gcore}^0\right] - \Pr\left[1 = (\mathcal{A} \to \mathcal{R}^{\text{ch-map}}) \to \mathtt{Gcore}^1(\mathcal{S}^{\text{core}})\right]\right| \\
= \;& \left|\Pr\left[1 = \mathcal{B} \to \mathtt{Gcore}^0\right] - \Pr\left[1 = \mathcal{B} \to \mathtt{Gcore}^1(\mathcal{S}^{\text{core}})\right]\right| \\
= \;& \mathsf{Adv}(\mathcal{B}, \mathtt{Gcore}^0, \mathtt{Gcore}^1) \\
= \;& \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\text{ch-map}}, \mathtt{Gcore}^0, \mathtt{Gcore}^1),
\end{aligned}$$

C.3 Proof of Lemma C.4 (Xpd-Inlining)

See Fig. 25c and Fig. 26a for the two games and the left and middle column of Fig. 29 for the code of `Map` and `Map-Xpd`, respectively. In Fig. 27, we depict the code of XPD_n of `Map` for $n \in O^*$ in the left-most column. From the left to the middle column, we inline the XPD_n code of the $\mathtt{Xpd}_n$ package, marked in pink. From the middle to the right column, we observe that n_1 and n_1' carry the same value, and that $label$ and $label'$ carry the same value. We thus just use n_1 and $label$ and remove the lines $n_1', \_ \leftarrow \mathsf{PrntN}(n)$ and $label' \leftarrow \mathsf{Labels}(n, r)$. Moreover, in the middle column, we branch on whether $n = psk$. Since $n \in O^*$ and $psk \notin O^*$, we always execute the **else** branch and can omit the branch for $n = psk$.

Map	Map → Xpd	Map-Xpd
$\mathsf{XPD}_{n\in O^*,\ell}(h_1, r, args)$	$\mathsf{XPD}_{n\in O^*,\ell}(h_1, r, args)$	$\mathsf{XPD}_{n\in O^*,\ell}(h_1, r, args)$
$i_1, \_ \leftarrow \mathsf{PrntIdx}(n, \ell)$	$i_1, \_ \leftarrow \mathsf{PrntIdx}(n, \ell)$	$i_1, \_ \leftarrow \mathsf{PrntIdx}(n, \ell)$
assert $M_{i_1}[h_1] \neq \bot$	**assert** $M_{i_1}[h_1] \neq \bot$	**assert** $M_{i_1}[h_1] \neq \bot$
$label \leftarrow \mathsf{Labels}(n, r)$	$label \leftarrow \mathsf{Labels}(n, r)$	$label \leftarrow \mathsf{Labels}(n, r)$
$\ell_1 \leftarrow \mathsf{level}(M_{i_1}[h_1])$	$\ell_1 \leftarrow \mathsf{level}(M_{i_1}[h_1])$	$\ell_1 \leftarrow \mathsf{level}(M_{i_1}[h_1])$
$h \leftarrow \mathsf{xpd}\langle n, label, h_1, args\rangle$	$h \leftarrow \mathsf{xpd}\langle n, label, h_1, args\rangle$	$h \leftarrow \mathsf{xpd}\langle n, label, h_1, args\rangle$
	$n_1, \_ \leftarrow \mathsf{PrntN}(n)$	$n_1, \_ \leftarrow \mathsf{PrntN}(n)$
	$label' \leftarrow \mathsf{Labels}(n, r)$	
$h' \leftarrow \mathsf{XPD}_{n,\ell_1}(M_{i_1}[h_1], r, args)$	$h' \leftarrow \mathsf{xpd}\langle n, label', M_{i_1}[h_1], args\rangle$	$h' \leftarrow \mathsf{xpd}\langle n, label, M_{i_1}[h_1], args\rangle$
	$(k_1, hon) \leftarrow \mathsf{GET}_{n_1,\ell_1}(M_{i_1}[h_1])$	$(k_1, hon) \leftarrow \mathsf{GET}_{n_1,\ell_1}(M_{i_1}[h_1])$
	if $n = psk$:	
	$\ell_1 \leftarrow \ell_1 + 1$	
	$k \leftarrow \mathsf{xpd}(k_1, (label, args))$	
	else	
	$d \leftarrow \mathsf{HASH}(args)$	$d \leftarrow \mathsf{HASH}(args)$
	$k \leftarrow \mathsf{xpd}(k_1, (label, d))$	$k \leftarrow \mathsf{xpd}(k_1, (label, d))$
	$() \leftarrow \mathsf{SET}_{n,\ell_1}(h', hon, k)$	$() \leftarrow \mathsf{SET}_{n,\ell_1}(h', hon, k)$
$M_n[h] \leftarrow h'$	$M_n[h] \leftarrow h'$	$M_n[h] \leftarrow h'$
return h	**return** h	**return** h

Fig. 27: Inlining proof (Lemma C.4)

C.4 Proof of Lemma C.5 (Map-Outro)

The proof proceeds by showing the following two claims.

Claim C.5.1 (Injectivity) *For all $n \in O^*$, the mapping $M_{n,\ell}$ is injective, i.e., if $M_{n,\ell}[h] = M_{n,\ell}[h']$, then $h = h'$.*

For the proof of this claim, we refer to Appendix C.5.

Map

$\mathsf{PrntIdx}(es, \ell)$

return $(0salt), (psk, \ell)$

$\mathsf{PrntIdx}(hs, \ell)$

return $(esalt, \ell), (dh)$

$\mathsf{PrntIdx}(as, \ell)$

return $(hsalt, \ell), (0ikm)$

$\mathsf{PrntIdx}(n, \ell)_{n\in XPN}$

$(n_1, \_) \leftarrow \mathsf{PrntN}(n)$
return $(n_1, \ell), ()$

Fig. 28: Index function of Map. See Fig. 29 for the code of the oracle of Map.

Map

$\mathsf{SET}_{\mathrm{psk},0}(h, hon, k)$

$M_{\mathrm{psk}}[h] \leftarrow \mathsf{SET}_{\mathrm{psk},0}(h, hon, k)$
return h

$\mathsf{DHGEN}()$

return $\mathsf{DHGEN}()$

$\mathsf{DHEXP}(X, Y)$

$h \leftarrow \mathsf{dh}\langle\mathsf{sort}(X, Y)\rangle$
$h' \leftarrow \mathsf{DHEXP}(X, Y)$
if $M_{dh}[h] = \bot$:
  $M_{dh}[h] \leftarrow h'$
return h

$\mathsf{XTR}_{n\in\{es,hs,as\},\ell}(h_1, h_2)$

$i_1, i_2 \leftarrow \mathsf{PrntIdx}(n, \ell)$
assert $M_{i_1}[h_1] \neq \bot$
assert $M_{i_2}[h_2] \neq \bot$
$\ell' \xleftarrow{\text{choose}} \mathsf{level}(M_{i_1}[h_1]), \mathsf{level}(M_{i_2}[h_2])$
$h \leftarrow \mathsf{xtr}\langle n, h_1, h_2\rangle$
$h' \leftarrow \mathsf{XTR}_{n,\ell'}(M_{i_1}[h_1], M_{i_2}[h_2])$
$M_{n,\ell}[h] \leftarrow h'$
return h

$\mathsf{XPD}_{n\in XPN,\ell}(h_1, r, args)$

$i_1, \_ \leftarrow \mathsf{PrntIdx}(n, \ell)$
assert $M_{i_1}[h_1] \neq \bot$
$label \leftarrow \mathsf{Labels}(n, r)$
$\ell_1 \leftarrow \mathsf{level}(M_{i_1}[h_1])$
$h \leftarrow \mathsf{xpd}\langle n, label, h_1, args\rangle$
$h' \leftarrow \mathsf{XPD}_{n,\ell_1}(M_{i_1}[h_1], r, args)$

if $n = psk : \ell \leftarrow \ell + 1$
$M_{n,\ell}[h] \leftarrow h'$
return h

$\mathsf{GET}_{n\in O^*,\ell}(h)$

assert $M_{n,\ell}[h] \neq \bot$
return $\mathsf{GET}_{n,\mathsf{level}(M_{n,\ell}[h])}(M_{n,\ell}[h])$

Map-Xpd

$\mathsf{SET}_{\mathrm{psk},0}(h, hon, k)$

$M_{\mathrm{psk}}[h] \leftarrow \mathsf{SET}_{\mathrm{psk},0}(h, hon, k)$
return h

$\mathsf{DHGEN}()$

return $\mathsf{DHGEN}()$

$\mathsf{DHEXP}(X, Y)$

Unchanged from `Map` on the left

$\mathsf{XTR}_{n\in\{es,hs,as\},\ell}(h_1, h_2)$

Unchanged from `Map` on the left

$\mathsf{XPD}_{n\in XPN\setminus O^*,\ell}(h_1, r, args)$

$i_1, \_ \leftarrow \mathsf{PrntIdx}(n, \ell)$
assert $M_{i_1}[h_1] \neq \bot$
$label \leftarrow \mathsf{Labels}(n, r)$
$\ell_1 \leftarrow \mathsf{level}(M_{i_1}[h_1])$
$h \leftarrow \mathsf{xpd}\langle n, label, h_1, args\rangle$
$h' \leftarrow \mathsf{XPD}_{n,\ell_1}(M_{i_1}[h_1], r, args)$
if $n = psk : \ell \leftarrow \ell + 1$
$M_{n,\ell}[h] \leftarrow h'$
return h

$\mathsf{XPD}_{n\in O^*,\ell}(h_1, r, args)$

$i_1, \_ \leftarrow \mathsf{PrntIdx}(n, \ell)$
assert $M_{i_1}[h_1] \neq \bot$
$label \leftarrow \mathsf{Labels}(n, r)$
$\ell_1 \leftarrow \mathsf{level}(M_{i_1}[h_1])$
$h \leftarrow \mathsf{xpd}\langle n, label, h_1, args\rangle$
$n_1, \_ \leftarrow \mathsf{PrntN}(n)$
$h' \leftarrow \mathsf{xpd}\langle n, label, M_{i_1}[h_1], args\rangle$
$(k_1, hon) \leftarrow \mathsf{GET}_{n_1,\ell_1}(M_{i_1}[h_1])$
$d \leftarrow \mathsf{HASH}(args)$
$k \leftarrow \mathsf{xpd}(k_1, (label, d))$
$() \leftarrow \mathsf{SET}_{n,\ell}(h, hon, k)$
$M_{n,\ell}[h] \leftarrow h'$
return h

$\mathsf{GET}_{n\in O^*,\ell}(h)$

assert $M_{n,\ell}[h] \neq \bot$
return $\mathsf{GET}_{n,\mathsf{level}(M_{n,\ell}[h])}(M_n[h])$

Map-Xpd-Remap

$\mathsf{SET}_{\mathrm{psk},0}(h, hon, k)$

$M_{\mathrm{psk}}[h] \leftarrow \mathsf{SET}_{\mathrm{psk},0}(h, hon, k)$
return h

$\mathsf{DHGEN}()$

return $\mathsf{DHGEN}()$

$\mathsf{DHEXP}(X, Y)$

Unchanged from `Map` on the left

$\mathsf{XTR}_{n\in\{es,hs,as\},\ell}(h_1, h_2)$

Unchanged from `Map` on the left

$\mathsf{XPD}_{n\in XPN\setminus O^*,\ell}(h_1, r, args)$

$i_1, \_ \leftarrow \mathsf{PrntIdx}(n, \ell)$
assert $M_{i_1}[h_1] \neq \bot$
$label \leftarrow \mathsf{Labels}(n, r)$
$\ell_1 \leftarrow \mathsf{level}(M_{i_1}[h_1])$
$h \leftarrow \mathsf{xpd}\langle n, label, h_1, args\rangle$
$h' \leftarrow \mathsf{XPD}_{n,\ell_1}(M_{i_1}[h_1], r, args)$
if $n = psk : \ell \leftarrow \ell + 1$
$M_{n,\ell}[h] \leftarrow h'$
return h

$\mathsf{XPD}_{n\in O^*,\ell}(h_1, r, args)$

$i_1, \_ \leftarrow \mathsf{PrntIdx}(n, \ell)$
assert $M_{i_1}[h_1] \neq \bot$
$label \leftarrow \mathsf{Labels}(n, r)$
$\ell_1 \leftarrow \mathsf{level}(M_{i_1}[h_1])$
$h \leftarrow \mathsf{xpd}\langle n, label, h_1, args\rangle$
$n_1, \_ \leftarrow \mathsf{PrntN}(n)$

$(k_1, hon) \leftarrow \mathsf{GET}_{n_1,\ell_1}(M_{i_1}[h_1])$
$d \leftarrow \mathsf{HASH}(args)$
$k \leftarrow \mathsf{xpd}(k_1, (label, d))$
$() \leftarrow \mathsf{SET}_{n,\ell}(h, hon, k)$
$M_{n,\ell}[h] \leftarrow h$
return h

Fig. 29: Oracles of `Map`, `Map-Xpd`, `Map-Xpd-Remap` for $\ell \in \{0 \dots d\}$. $\ell' \xleftarrow{\text{choose}} \mathsf{level}(M_{n_1}[h_1]), \mathsf{level}(M_{n_2}[h_2])$ assigns to ℓ' the value $\mathsf{level}(M_{n_1}[h_1])$ if it is not $\bot$ and $\mathsf{level}(M_{n_2}[h_2])$, else.

Claim C.5.2 (Functional Equivalence) *If for all $n \in O^*$, the mapping $M_{n,\ell}$ is injective, then* $\texttt{Gks}^{Mapxpd} \overset{func}{\equiv} \texttt{Gks}^1(\mathcal{S})$

We prove Claim C.5.2 using injective variable renaming in the $\texttt{Key}_{O^*,0..d}$ packages. Namely, for all $n \in O^*$, $\ell \in \{0,\ldots,d\}$, we rename $M_{n,\ell}[h]$ to h. The renaming is well-defined for all values in the range of $M_{n,\ell}$, since there are no two distinct h and h' with $M_{n,\ell}[h] = M_{n,\ell}[h']$ by Claim C.5.1. In addition, the condition **assert** $M_{n,\ell}[h] \neq \bot$ in `Map` (on the left) and **assert** $K_{n,\ell}[h] \neq \bot$ (on the right) in `Key` are satisfied for the same handles h.

To show that this is indeed true, we now provide a detailed invariant argument, analogous to the proof of Lemma C.2. Namely, we think as *right* of the state of $\texttt{Gks}^{\text{Mapxpd}}$ (and add a superscript right) and as *left* of the state of $\texttt{Gks}^1(\mathcal{S})$ (and add a superscript left). Now, consider the following relations:

(0) $$E^{\text{left}} = E^{\text{right}}$$

(1) $$\forall n \in \{dh, 0ikm, 0salt\} : M_n^{\text{left}} = M_n^{\text{right}}$$
$$K_n^{\text{left}} = K_n^{\text{right}}$$
$$Log_n^{\text{left}} = Log_n^{\text{right}}$$

(2) $$\forall \ell \in \{0..d\}, n \in N \setminus (O^* \cup \{dh, 0ikm, 0salt\}) : M_{n,\ell}^{\text{left}} = M_{n,\ell}^{\text{right}}$$
$$K_{n,\ell}^{\text{left}} = K_{n,\ell}^{\text{right}}$$
$$Log_n^{\text{left}} = Log_n^{\text{right}}$$

(3) $$\forall \ell \in \{0..d\}, n \in O^* : M_{n,\ell}^{\text{left}}[h] = \bot \Leftrightarrow K_{n,\ell}^{\text{right}}[h] = \bot$$
$$K_{n,\ell'}^{\text{left}}[M_{n,\ell}^{\text{left}}[h]] = K_{n,\ell}^{\text{right}}[h]$$
$$\text{for } \ell' = \mathsf{level}(M_{n,\ell}^{\text{left}}[h])$$
$$Log_n^{\text{left}}[M_{n,\ell}^{\text{left}}[h]] = (M_{n,\ell}^{\text{left}}[h], hon, k) \Leftrightarrow Log_n^{\text{right}}[h] = (h, hon, k)$$

(0) holds since only the GENDH query of `DH` writes to E (a) on the same inputs and (b) without reading any state.

(1) holds since M_n, K_n and Log_n, for $n \in \{0salt, 0ikm\}$ are initialized with the same values and never modified/written to. For $n = dh$, only the DHEXP oracle makes SET_{dh} queries which write to M_{dh} and K_{dh}. The code of the DHEXP oracle remains unchanged from $\texttt{Gks}^{\text{Mapxpd}}$ to $\texttt{Gks}^1(\mathcal{S})$, and it only depends on state which is equal, namely the table E, which we discussed in (0).

(2) For $n \in N \setminus (O^* \cup \{dh, 0ikm, 0salt\})$, only $\mathsf{XPD}_{n,*}$ or $\mathsf{XTR}_{n,*}$ queries write to $M_{n,*}$, $K_{n,*}$ and Log_n. They have identical code in $\texttt{Gks}^{\text{Mapxpd}}$ and $\texttt{Gks}^1(\mathcal{S})$ and only read state which is equal (by induction hypothesis). Thus, the levels of the tables which are modified are also identical.

(3) Let us now consider how the oracles $\mathsf{XPD}_{n,\ell}$ and $\mathsf{GET}_{n,\ell}$ with $n \in O^*$ affect condition (3). $\mathsf{GET}_{n,*}$ only performs read operations and thus does not modify $M_{n,*}$, $K_{n,*}$ and Log_n. Before turning to the induction step, notice that it suffices to prove line 2 of condition (3). The reason is that if $K_{n,\ell'}^{\text{left}}[M_{n,\ell}^{\text{left}}[h]] \neq \bot$, then $Log_n^{\text{left}}[M_{n,\ell}^{\text{left}}[h]] = (M_{n,\ell}^{\text{left}}[h], hon, k)$, borrowing condition (2a) (iv) from the

proof of Lemma C.2 (since that part of the proof is local over `Key` and `Log`). The same statement holds on the right, and therefore, using line 2 of condition (3), the *hon* and k values in the *Log* are equal, too, i.e., $Log_n^{\text{left}}[M_{n,\ell}^{\text{left}}[h]] = (M_{n,\ell}^{\text{left}}[h], hon, k) \Leftrightarrow Log_n^{\text{right}}[h] = (h, hon, k)$.

We now assume that (1), (2) and (3) hold and prove via induction over $\mathsf{XPD}_{n,\ell}$ for $n \in O^*$ that line 2 of (3) holds after calling $\mathsf{XPD}_{n,\ell}$. Consider the $\mathsf{XPD}_{n,\ell}$ oracle in $\texttt{Gks}^{\text{Mapxpd}}$. If $M_{n,\ell}^{\text{left}}[h] = \bot$ and $\mathsf{XPD}_{n,\ell}$ writes into $M_{n,\ell}^{\text{left}}[h]$, then the $\mathsf{XPD}_{n,\ell}$ oracle in $\texttt{Gks}^1(\mathcal{S})$ writes into $K_{n,\ell}^{\text{right}}[h]$, since by induction assumption, $K_{n,\ell}^{\text{right}}[h]$ has not been written to before. Moreover, by injectivity of the mapping, $K_{n,\ell'}^{\text{left}}[M_{n,\ell}^{\text{left}}[h]]$ with $\ell' = \mathsf{level}(M_{n,\ell}^{\text{left}}[h])$ has not been written to before (since there is no other handle h' with $M_{n,*}^{\text{left}}[h'] = M_{n,\ell}^{\text{left}}[h]$) and thus receives the same value k as $K_{n,\ell}^{\text{right}}[h]$, which concludes the proof of Claim C.5.2.

C.5 Injectivity

Claim C.5.1 (Injectivity) *For all $n \in O^*$, the mapping $M_{n,\ell}$ is injective, i.e., if $M_{n,\ell}[h] = M_{n,\ell}[h']$, then $h = h'$.*

Proof. The proof proceeds by induction over oracle calls to the `Map-XPD` package. We use the notation $M_{n,*}$ to denote $M_{n,\ell}$ for some arbitrary ℓ and by overloading of notation to also denote M_n for $n \in \{\mathit{0salt}, \mathit{dh}, \mathit{0ikm}\}$. $\mathsf{psk}(h)$ is a helper function which takes as input handle h and deconstructs it until finding a *psk* handle (which will have the same level as h). Similarly, $\mathsf{dh}(h)$ deconstructs h until finding a *dh* handle (which has no level). $\mathsf{rm}(h)$ reconstructs h until finding a *rm* handle (which will have one level less than h). We use similar helper functions for further key names when needed. In the following invariant, we use (but omit to state) (2a) (iv) and (2d) which were stated in the invariant of Lemma C.2 and can be proven analogously.

Invariant

(1) Arguments: If $M_{n,*}[h] = M_{n,*}[h'] = h''$, then recursively on the handle structure, $\mathit{args}(h) = \mathit{args}(h') = \mathit{args}(h'')$ (including the labels), recursing until one of the handles reaches a level 0 psk handle.

(2) Binder property: level 0 and level > 0 binders are different, i.e., for all handles h and h' with $\mathsf{name}(h) = \mathsf{name}(h') = \mathit{binder}$, $\mathsf{level}(h) = 0$ and $\mathsf{level}(h') := \ell' > 0$, we have that

$$K_{\mathit{binder},\mathsf{level}(M_{\mathit{binder},0}[h])}[M_{\mathit{binder},0}[h]] \neq K_{\mathit{binder},\mathsf{level}(M_{\mathit{binder},\ell'}[h'])}[M_{\mathit{binder},\ell'}[h']].$$

(3a) PSK mapping:

$$M_{n,*}[h] = M_{n,*}[h'] \wedge h \neq h' \wedge \mathsf{psk}(h) \neq \mathsf{psk}(h') \Rightarrow \mathsf{level}(h) \neq \mathsf{level}(h') \wedge 0 \in \{\mathsf{level}(h), \mathsf{level}(h')\}$$

(3b) DH mapping:

$$M_{n,*}[h] = M_{n,*}[h'] \wedge h \neq h' \wedge \mathsf{psk}(h) = \mathsf{psk}(h') \Rightarrow \mathsf{level}(h) = \mathsf{level}(h') \wedge \mathsf{dh}(h) \neq \mathsf{dh}(h')$$

(4) Injectivity:
$$\forall n \in N^{S\leq}, h, h' : M_{n,*}[h] = M_{n,*}[h'] \neq \bot \Rightarrow h = h',$$
where $N^{S\leq}$ contains S and its decendants within the same level (no psk).

Assume (1)–(4) hold before an oracle call. We show that (1)–(4) still holds afterwards.

Property (1): If $M_{n,*}[h] = h'$, then the handles $M_{n,*}[h]$ and h' match recursively on their *args* and labels. Below, we see that the matching is either propagated, or irrelevant (since there are no *args*).

$\mathsf{SET}_{psk,0}$**:** This oracle may only be called for $\ell = 0$ and h has therefore to be of the structure $\mathsf{psk}\langle ctr, alg\rangle$ (otherwise, the $\mathtt{Key}_{psk,0}$ package aborts in an assert) which does not contain any *args*.

DHGEN**:** This oracle does not use handles.

DHEXP**:** h and h' are of the form $\mathsf{dh}\langle X, Y\rangle$ for some X, Y which does not contain any *args*.

$\mathsf{XTR}_{n,\ell}$**:** Both h and h' do not add new *args* and the condition already holds for the included handles.

$\mathsf{XPD}_{n,\ell}$**:** The *args* (including the labels) added to h and h' are consistently used in the `Map` and `Xpd` packages.

Property (2):

$\mathsf{XPD}_{binder,\ell}$**:** Let h be the response to the oracle call and $h' \in M_{binder,*} \neq h$ with $\ell = \mathsf{level}(h) \neq \mathsf{level}(h') = \ell'$ and $0 \in \{\mathsf{level}(h), \mathsf{level}(h')\}$. Observe that $\mathsf{Label}(binder, r)$ is different for $r = 0$ and $r = 1$ and, as `Check` enforces $r = 0$ for level 0 and $r = 1$ otherwise, $\mathsf{label}(h) \neq \mathsf{label}(h)$.
Moreover, due to (1), the labels in $\tilde{h} := M_{binder,*}[h]$ and $\tilde{h'} := M_{binder,*}[h']$ are different, too and the two binder values are stored in $K_{binder,*}[\tilde{h}]$ and $K_{binder,*}[\tilde{h'}]$, respectively. Since $\mathtt{Log}_{binder}$ uses an F-pattern, values do not repeat and thus, $K_{binder,*}[\tilde{h}]$ are $K_{binder,*}[\tilde{h'}]$ are different as required.

Otherwise: $M_{binder,*}$ is not modified.

Property (3a):

$\mathsf{SET}_{psk,0}(h, hon, k)$**:** W.l.o.g. $\mathtt{Log}_{psk,0}[h] = \bot$ before the call since else the state is not modified. $\mathtt{Log}_{psk,0}[h] = \bot$ implies that $K_{psk,0}[h] = \bot$ before the call (via a local argument over `Log` and `Key` which we explicitly stated in 2a (iv) in the proof of Lemma C.2).

Case I There exists $h'' \neq h$ with $\mathsf{level}(h'') = 0$ and $Log[h''] = (*, 0, k)$: The A pattern aborts now.

Case II There exists no $h'' \neq h$ with $\mathsf{level}(h'') = 0$ and $Log[h''] = (*, 0, k)$: The A pattern does not abort and $J[k] = \bot$ before the call.

(a) There is a $h' \neq h$ with $\mathsf{level}(h') \neq 0$ und $Log[h'] = (*, 0, k)$. The game sets $J[k] \leftarrow 1$ and condition (3a) continues to hold.

(b) There is no $h' \neq h$ with $\mathsf{level}(h') \neq 0$ und $Log[h'] = (*, 0, k)$: No new mapping occurs and thus (3a) continues to hold.

$\mathsf{XPD}_{psk,\ell}(h_{rm}, r, args)$**:** Let h be the response from the oracle call. Observe that $\mathsf{level}(h) \neq 0$. If there is no $h' \neq h$ with $M_{psk,*}[h] = M_{psk,*}[h']$, condition (3a) holds. In the other case, let h' be such that $h' \neq h$ with $M_{psk,*}[h] = M_{psk,*}[h']$.

Case I $\mathsf{level}(h') = 0$, (3a) holds.

Case II $\mathsf{level}(h') \neq 0$.

(a) $args = \mathsf{args}(h')$. In this case, since the resumption bit is 1 for both, and both have the same *args*, the only value which can be different and lead to $h \neq h'$ is the *rm* handle, i.e., $\mathsf{rm}(h) \neq \mathsf{rm}(h')$. By (2d) in Lemma C.2, since $M_{psk,*}[h] = M_{psk,*}[h']$, we also have that $M_{rm,*}[Orm(h)] = M_{rm,*}[\mathsf{rm}(h')]$. Now, due to (4), this implies that $\mathsf{rm}(h) = \mathsf{rm}(h')$ in contradiction to $\mathsf{rm}(h) \neq \mathsf{rm}(h')$. Thus, this case is impossible.

(b) $args \neq \mathsf{args}(h')$. This is impossible as (1) ensures recursive consistency on the arguments.

$\mathsf{XPD}_{n,\ell}(h_1, r, args), n \neq psk$**:** Let h be the response from the oracle call. Observe that `Key.SET` will either aborts or returns the handle unmodified. If there exists $h' \neq h$ such that $M_n[h] = M_n[h']$ then $\mathsf{args}(h) = \mathsf{args}(h')$ and $\mathsf{parent}(h) = \mathsf{parent}(h')$ (code of map package) Due to (3a) applied to the parents and $\mathsf{level}(h) = \mathsf{level}(\mathsf{parent}(h))$ and $\mathsf{level}(h') = \mathsf{level}(\mathsf{parent}(h'))$ (3a) also holds for h and h'.

$\mathsf{XTR}_{n,\ell}(h_1, h_2)$ Observe that (3a) follows from an analogous argument for $\mathsf{XTR}_{es,\ell}$ and $\mathsf{XTR}_{as,\ell}$. For $\mathsf{XTR}_{hs,\ell}$ the argument follows from applying (3a) to the *esalt* parent handles.

DHEXP**:** Let h be the response from the oracle call. Let $h' \neq h$ with $M_{psk}[h] = M_{psk}[h']$. h and h' are of the form $\mathsf{dh}\langle X, Y\rangle$ for some X, Y and therefore $\mathsf{psk}(h) = \mathsf{psk}(h') = \perp$ in contradiction to $h' \neq h$ and therefore (3a) holds.

Property (3b):

$\mathsf{XTR}_{hs,\ell}(h_{esalt}, h_{dh})$**:** Let h be the response from the oracle call. Let $h' \neq h$ with $M_{hs,*}[h] = M_{hs,*}[h']$ and $\mathsf{psk}(h) = \mathsf{psk}(h')$. By (1) $\mathsf{args}(h) = \mathsf{args}(h')$ and therefore $\mathsf{esalt}(h) = \mathsf{esalt}(h')$. Since $h' \neq h$ it follows that $\mathsf{dh}(h) \neq \mathsf{dh}(h')$ and therefore (3b).

Otherwise: Since the psk is equal, both handles indeed have the same level. The inequality of the dh shares follows from using (3b) on the parent handles and the close relation between parents and children ((2d) from Lemma C.2).

Property (4):

$\mathsf{XPD}_{n,\ell}(h_1, r, args)$ We first consider $n \in S$ since this is the most intricate case. Let h be the response from the oracle call. Towards contradiction, let $h' \neq h$ with $M_n[h] = M_n[h']$. Observe that $\mathsf{args}(h) = \mathsf{args}(h')$ due to (1).

Case I $\mathsf{psk}(h) = \mathsf{psk}(h')$ then by (3b) $\mathsf{dh}(h) \neq \mathsf{dh}(h')$. Observe that Check ensures that $\mathsf{DHArgs}(args) \neq \mathsf{DHArgs}(args')$ and therefore, it follows that $\mathsf{args}(h) \neq \mathsf{args}(h')$ in contradiction to the observation $\mathsf{args}(h) = \mathsf{args}(h')$.

Case II $\mathsf{psk}(h) \neq \mathsf{psk}(h')$ then due to (3a) $\mathsf{level}(h) \neq \mathsf{level}(h')$ and $0 \in \{\mathsf{level}(h), \mathsf{level}(h')\}$.

(a) $\mathsf{Label}(n,0) \neq \mathsf{Label}(n,1)$ in contradiction to $\mathsf{args}(h) = \mathsf{args}(h')$.
(b) Let $h_{binder} := \mathsf{BinderHand}(h, \mathsf{args}(h))$, and we denote $h'_{binder} := \mathsf{BinderHand}(h', \mathsf{args}(h'))$. As $\mathsf{level}(h_{binder}) = \mathsf{level}(h)$, $\mathsf{level}(h'_{binder}) = \mathsf{level}(h')$, $\mathsf{level}(h_{binder}) \neq \mathsf{level}(h'_{binder})$ and 0 is contained in the set $\{\mathsf{level}(h_{binder}), \mathsf{level}(h'_{binder})\}$, due to (2), we have that $k := K_{binder,0}[M_{binder}[h_{binder}]]$ and $k' := K_{binder,\ell'}[M_{binder,*}[h'_{binder}]]$ are not equal. `Check` ensures that $k = \mathsf{BinderArgs}(\mathsf{args}(h))$ and $k' = \mathsf{BinderArgs}(\mathsf{args}(h'))$ and therefore $\mathsf{args}(h) \neq \mathsf{args}(h')$ in contradiction to $\mathsf{args}(h) = \mathsf{args}(h')$.

We now consider n such that n comes *after* a separation point on the path from *psk*, then (4) follows directly from (4) on the parents and using the relation to the parent handles via (2d) of Lemma C.2. In turn if n comes *before* a separation point on the path from *psk*, there is no write on $M_{n,*}$.

D Core Key Schedule Theorem

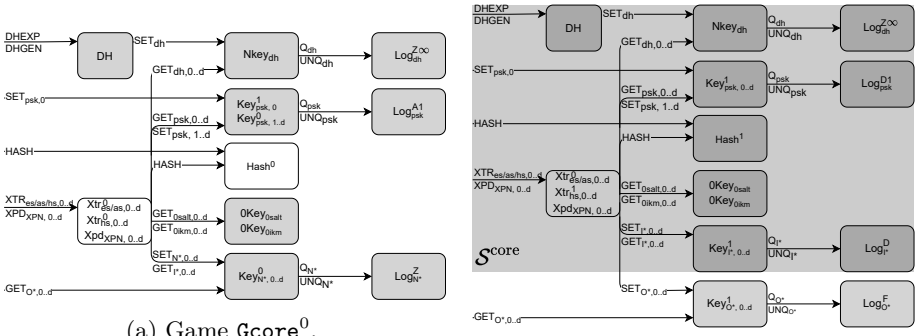
The proof of Theorem D.1 is outlined in Fig. 31. The core step of the proof (Lemma D.6) is a two-dimensional hybrid argument outlined in Fig. 32. The first hybrid argument is over resumption levels and follows a standard pattern. The inner hybrid argument then proceeds over the graph structure of the key schedule. This is where the generalized statement of a key schedule is convenient: While we argue in Lemmas D.11, D.12, D.13 explicitely about the `xtr` assumptions, reasoning over all `xpd` assumptions proceeds by a generic argument (Lemma D.14) and is independent of the actualy set of `xpd` packages and the number of keys derived.

Before we can proof this lemma, we need a sequence of 4 game hops. In the first step (Lemma D.2) we idealize collission resistence of the `Hash` package (which is used to hash transcripts). The second step (Lemma D.3) changes the pattern in the `Log` packages from Z to D. The proof of Lemma D.3 reduces to collision-resistance and also relies on an *inductive* argument called the *co-dependance lemma*. Namely, we show that once collision-resistance of the hash-function is idealized, a D abort on one key requires that there has been a collision before and therefore, since every collision requires a previous collision, no collision can occur. In the third step (Lemma D.4) we add an *rewarding abort* on the *esalt* key where we return a special winning symbol to the adversary (which allows the adversary to trivially distinguish) if the adversary can cause a collision containing an honest *esalt* key. This can only increase an adversary's distinguishing probability, so, it suffices to show that we can bound the resulting advantage. With the collision-freeness of *esalt* we can (globally) apply the `Gsodh` assumption to idealize Diffie-Hellman secrets (Lemma D.5). At this point we can idealize the `xtr`and `xpd`assumptions using the central hybrid argument (Lemma D.6). Finally we use preimage-resistance (Lemma D.7) to remove the rewarding aborts introduced in Lemma D.4. We then observe that the game (Fig. 31h) can be split in simulator and ideal functionality as desired.

Theorem D.1 (Core). *Let ks be a TLS-like key schedule with XPR. Let d be an integer. Let $\mathcal{S}^{core}$ be the efficient simulator defined in Fig. 30a. Then, for all adversaries $\mathcal{A}$ which make queries for at most d resumption levels, we have that*

$$\begin{aligned}
&\mathsf{Adv}(\mathcal{A}, \mathtt{Gcore}^0, \mathtt{Gcore}^1(\mathcal{S}^{core})) \\
\leq &\sum_{\mathcal{R}\in\{\mathcal{R}_{cr},\mathcal{R}_Z,\mathcal{R}_D\}} \mathsf{Adv}(\mathcal{A}\to\mathcal{R}, \mathtt{Gacr}^b) + \max_{i\in\{0,1\}} \mathsf{Adv}(\mathcal{A}_i\to\mathcal{R}_{sodh}, \mathtt{Gsodh}^b) \\
&+ \mathsf{Adv}(\mathcal{A}_i\to\mathcal{R}_{esalt,pi}, \mathtt{Gpi}^b_{esalt}) + \mathsf{Adv}(\mathcal{A}_i\to\mathcal{R}_{O^*,pi}, \mathtt{Gpi}^b_{O^*}) + \\
&\sum_{\ell=0}^{d-1} \big(\mathsf{Adv}(\mathcal{A}_i\to\mathcal{R}_{es,\ell}, \mathtt{Gxtr}^b_{es,\ell}) + \mathsf{Adv}(\mathcal{A}_i\to\mathcal{R}_{hs,\ell}, \mathtt{Gxtr}^b_{hs,\ell}) + \\
&\qquad \mathsf{Adv}(\mathcal{A}_i\to\mathcal{R}_{as}, \mathtt{Gxtr}^b_{as,\ell}) + \sum_{n\in XPR} \big(\mathsf{Adv}(\mathcal{A}_i\to\mathcal{R}_{n,\ell}, \mathtt{Gxpd}^b_{n,\ell})\big)\big),
\end{aligned}$$

where XPR is the required representation set, $\mathtt{Gcore}^0$ *and* $\mathtt{Gcore}^1(\mathcal{S}^{core})$ *are defined in Fig. 30,* $\mathcal{R}_{cr}$ *is defined in Fig. 31a,* $\mathcal{R}_{sodh}$ *is defined in Fig. 32a,* $\mathcal{R}_{es,\ell}$ *is defined in Fig. 34a,* $\mathcal{R}_{hs,\ell}$ *and* $\mathcal{R}_{as,\ell}$ *are defined analogously, and* $\mathcal{R}_{n,\ell}$ *for* $n \in XPR$ *and* $0 \leq \ell \leq d$ *is defined in Fig. 34b,* $\mathcal{R}_{esalt,pi}$ *is defined in Fig. 32c and* $\mathcal{R}_{O^*,pi}$ *is defined in Fig. 32d.*



(a) Game $\mathtt{Gcore}^0$.

(b) Game $\mathtt{Gcore}^1(\mathcal{S}^{\text{core}})$, $\mathcal{S}^{\text{core}}$ in grey.

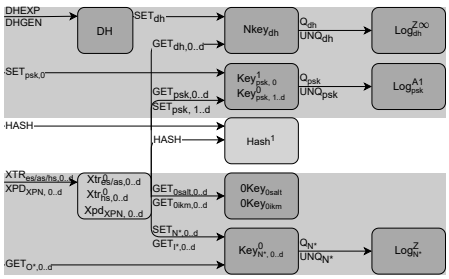
Fig. 30: Games for Theorem D.1

D.1 Proof of Theorem D.1

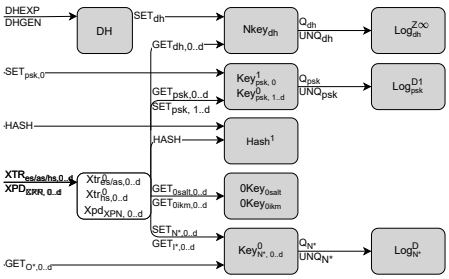
Lemma D.2 (Collision-Resistance).

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gcore}^0, \mathtt{Gcore}^{\mathsf{Hash}}) \leq \mathsf{Adv}(\mathcal{A}\to\mathcal{R}_{cr}, \mathtt{Gacr}^{\mathsf{hash},b}),$$

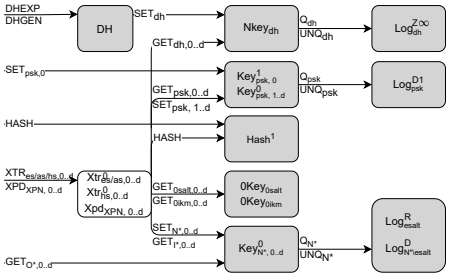
where $\mathtt{Gcore}^{\mathsf{Hash}}$ *is defined in Fig. 31a, and* $\mathcal{R}_{cr}$ *is marked in grey in Fig. 31a.*



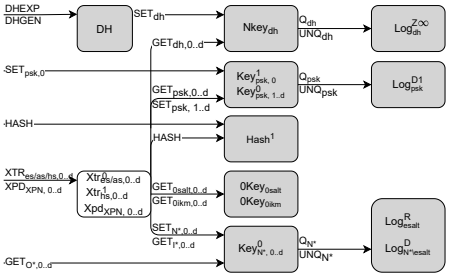
(a) Game $\texttt{Gcore}^{\mathsf{Hash}}$.



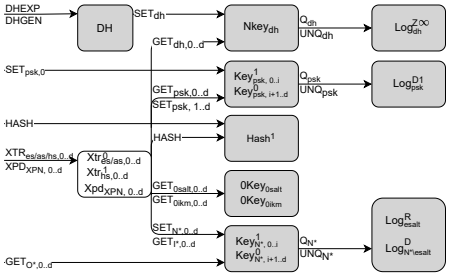
(b) Game $\texttt{Gcore}^{D}$



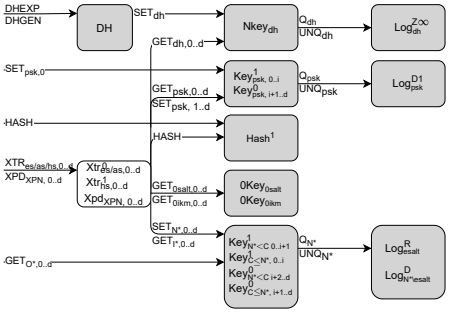
(c) Game $\texttt{Gcore}^{\mathsf{Resalt}}$.



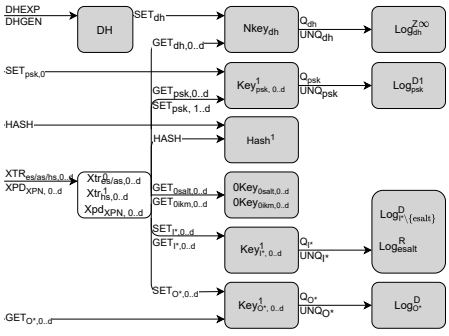
(d) Game $\texttt{Gcore}^{\mathsf{SODH}}$



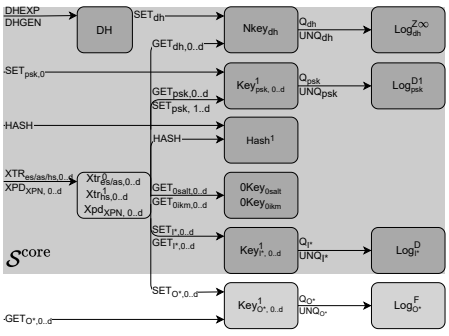
(e) Game $\texttt{Gcore}^{\mathrm{hyb}:\ell}$



(f) Game $\texttt{Gcore}^{\mathrm{hyb}:\ell-1,c}$



(g) Game $\texttt{Gcore}^{\mathrm{ki}}$



(h) Game $\texttt{Gcore}^{1}(\mathcal{S}^{\mathrm{core}})$, $\mathcal{S}^{\mathrm{core}}$ in grey.

Fig. 31: Proof overview for Theorem D.1

Lemma D.3 (D-Pattern).

$$\begin{aligned}&\mathsf{Adv}(\mathcal{A}, \mathtt{Gcore}^{\mathsf{Hash}}, \mathtt{Gcore}^{D}) \leq \\ &\mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{Z,\mathsf{xtr}}, \mathtt{Gacr}^{\mathsf{xtr},b}) + \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{Z,\mathsf{xpd}}, \mathtt{Gacr}^{\mathsf{xpd},b}) + \\ &\mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{D,\mathsf{xtr}}, \mathtt{Gacr}^{\mathsf{xtr},b}) + \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{D,\mathsf{xpd}}, \mathtt{Gacr}^{\mathsf{xpd},b}),\end{aligned}$$

where $\mathtt{Gcore}^{\mathsf{Hash}}$ *is defined in Fig. 31a,* $\mathtt{Gcore}^{D}$ *is defined in Fig. 31b,* $\mathcal{R}_{Z,\mathsf{xtr}}$ *is marked in grey in Fig. 33a,* $\mathcal{R}_{Z,\mathsf{xpd}}$ *is marked in grey in Fig. 33b,* $\mathcal{R}_{D,\mathsf{xpd}}$ *is marked in grey in Fig. 33c, and* $\mathcal{R}_{D,\mathsf{xtr}}$ *is marked in grey in Fig. 33d.*

Lemma D.4 (Resalt). *For every adversary* $\mathcal{A}$*, there exists two adversaries* $\mathcal{A}_0$ *and* $\mathcal{A}_1$ *of essentially the same runtime such that*

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gcore}^{\mathsf{Resalt}}, \mathtt{Gcore}^{1}(\mathcal{S}^{core})) \leq \max_{i \in \{0,1\}} \mathsf{Adv}(\mathcal{A}_i, \mathtt{Gcore}^{\mathsf{Hash}}, \mathtt{Gcore}^{D}),$$

where $\mathtt{Gcore}^{Resalt}$ *is defined in Fig. 31b, and for* $i \in \{0,1\}$*,* $\mathcal{A}_i$ *behaves as* $\mathcal{A}$ *except that it outputs bit* i *when receiving a win abort value.*

Lemma D.5 (SODH Lemma).

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gcore}^{\mathsf{Resalt}}, \mathtt{Gcore}^{\mathsf{SODH}}) = \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{sodh}, \mathtt{Gsodh}^{b}),$$

where $\mathtt{Gcore}^{\mathsf{SODH}}$ *is defined in Fig. 31c, and* $\mathcal{R}_{sodh}$ *is marked in grey in Fig. 32a.*

Lemma D.6 (Hybrid Lemma). *For all* $\ell = 0..d-1$*,*

$$\begin{aligned}&\mathsf{Adv}(\mathcal{A}, \mathtt{Gcore}^{hyb:\ell}, \mathtt{Gcore}^{hyb:\ell+1}) \\ \leq &\mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{es,\ell}, \mathtt{Gxtr}^{b}_{es,\ell}) + \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{hs,\ell}, \mathtt{Gxtr}^{b}_{hs,\ell}) \\ &+ \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{as,\ell}, \mathtt{Gxtr}^{b}_{as,\ell}) + \sum_{n \in XPR} \big(\mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{n,\ell}, \mathtt{Gxpd}^{b}_{n,\ell})\big),\end{aligned}$$

where $\mathtt{Gcore}^{hyb:\ell}$ *is defined in Fig. 31e.*

Lemma D.7 (Pre-image-resistance).

$$\begin{aligned}&\mathsf{Adv}(\mathcal{A}, \mathtt{Gcore}^{ki}, \mathtt{Gcore}^{1}(\mathcal{S}^{core})) \\ \leq &\mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{esalt,pi}, \mathtt{Gpi}^{R}_{esalt}, \mathtt{Gpi}^{D}_{esalt}) + \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{O^*,pi}, \mathtt{Gpi}^{D}_{O^*}, \mathtt{Gpi}^{F}_{O^*}),\end{aligned}$$

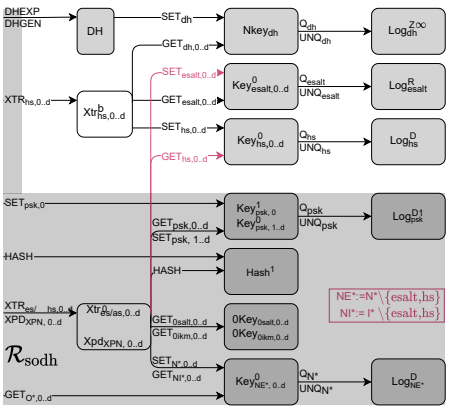
where $\mathcal{R}_{esalt,pi}$ *is defined in Fig. 32c and* $\mathcal{R}_{O^*,pi}$ *is defined in Fig. 32d.*

From Lemma D.2-D.7, we obtain Theorem D.1 by observing that

$$\mathtt{Gcore}^{\mathsf{SODH}} = \mathtt{Gcore}^{\mathrm{hyb:0}} \text{ and } \mathtt{Gcore}^{\mathrm{ki}} = \mathtt{Gcore}^{\mathrm{hyb:d}} \tag{20}$$

and then applying a standard hybrid argument, included here for completeness:

$$
\begin{aligned}
&\mathsf{Adv}(\mathcal{A}, \mathtt{Gcore}^0, \mathtt{Gcore}^1(\mathcal{S}_{\mathrm{core}}))\\
\overset{\text{Lm. } D.2}{\leq}\ &\mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{\mathrm{cr}}, \mathtt{Gacr}^b) + \mathsf{Adv}(\mathcal{A}, \mathtt{Gks}^{\mathsf{Hash}}, \mathtt{Gcore}^1(\mathcal{S}^{\mathrm{core}}))\\
\overset{\text{Lm. } D.3}{\leq}\ &\sum_{\mathcal{R}\in\{\mathcal{R}_{\mathrm{cr}},\mathcal{R}_Z,\mathcal{R}_D\}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}, \mathtt{Gacr}^b) + \mathsf{Adv}(\mathcal{A}, \mathtt{Gcore}^{\mathsf{D}}, \mathtt{Gcore}^1(\mathcal{S}^{\mathrm{core}}))\\
\overset{\text{Lm. } D.4}{\leq}\ &\sum_{\mathcal{R}\in\{\mathcal{R}_{\mathrm{cr}},\mathcal{R}_Z,\mathcal{R}_D\}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}, \mathtt{Gacr}^b) + \max_{i\in\{0,1\}} \mathsf{Adv}(\mathcal{A}_i, \mathtt{Gcore}^{\mathsf{Resalt}}, \mathtt{Gcore}^1(\mathcal{S}^{\mathrm{core}}))\\
\overset{\text{Lm. } D.5}{=}\ &\sum_{\mathcal{R}\in\{\mathcal{R}_{\mathrm{cr}},\mathcal{R}_Z,\mathcal{R}_D\}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}, \mathtt{Gacr}^b) + \max_{i\in\{0,1\}} \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}_{\mathrm{sodh}}, \mathtt{Gsodh}^b)\\
&+ \mathsf{Adv}(\mathcal{A}_i, \mathtt{Gcore}^{\mathsf{SODH}}, \mathtt{Gcore}^1(\mathcal{S}^{\mathrm{core}}))\\
\overset{\text{Eq. } 20}{\leq}\ &\sum_{\mathcal{R}\in\{\mathcal{R}_{\mathrm{cr}},\mathcal{R}_Z,\mathcal{R}_D\}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}, \mathtt{Gacr}^b) + \max_{i\in\{0,1\}} \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}_{\mathrm{sodh}}, \mathtt{Gsodh}^b)\\
&+ \mathsf{Adv}(\mathcal{A}_i, \mathtt{Gcore}^{\mathrm{ki}}, \mathtt{Gcore}^1(\mathcal{S}^{\mathrm{core}})) + \mathsf{Adv}(\mathcal{A}_i, \mathtt{Gcore}^{\mathrm{hyb:0}}, \mathtt{Gcore}^{\mathrm{hyb:d}})\\
\overset{\text{tele. sum}}{\leq}\ &\sum_{\mathcal{R}\in\{\mathcal{R}_{\mathrm{cr}},\mathcal{R}_Z,\mathcal{R}_D\}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}, \mathtt{Gacr}^b) + \max_{i\in\{0,1\}} \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}_{\mathrm{sodh}}, \mathtt{Gsodh}^b)\\
&+ \mathsf{Adv}(\mathcal{A}_i, \mathtt{Gcore}^{\mathrm{ki}}, \mathtt{Gcore}^1(\mathcal{S}^{\mathrm{core}})) + \sum_{\ell=0}^{d-1} \mathsf{Adv}(\mathcal{A}_i, \mathtt{Gcore}^{\mathrm{hyb:}\ell}, \mathtt{Gcore}^{\mathrm{hyb:}\ell+1})\\
\overset{\text{Lm. } D.6}{\leq}\ &\sum_{\mathcal{R}\in\{\mathcal{R}_{\mathrm{cr}},\mathcal{R}_Z,\mathcal{R}_D\}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}, \mathtt{Gacr}^b) + \max_{i\in\{0,1\}} \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}_{\mathrm{sodh}}, \mathtt{Gsodh}^b)\\
&+ \mathsf{Adv}(\mathcal{A}_i, \mathtt{Gcore}^{\mathrm{ki}}, \mathtt{Gcore}^1(\mathcal{S}^{\mathrm{core}}))+\\
&\sum_{\ell=0}^{d-1} \big(\mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}_{\mathrm{es},\ell}, \mathtt{Gxtr}^b_{es,\ell}) + \max_{i\in\{0,1\}} \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}_{\mathrm{hs},\ell}, \mathtt{Gxtr}^b_{hs,\ell})+\\
&\qquad \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}_{\mathrm{as}}, \mathtt{Gxtr}^b_{as,\ell}) + \sum_{n\in XPR} \big(\mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}_{\mathrm{n},\ell}, \mathtt{Gxpd}^b_{n,\ell})\big)\big)\\
\overset{\text{Lm. } D.7}{\leq}\ &\sum_{\mathcal{R}\in\{\mathcal{R}_{\mathrm{cr}},\mathcal{R}_Z,\mathcal{R}_D\}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}, \mathtt{Gacr}^b) + \max_{i\in\{0,1\}} \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}_{\mathrm{sodh}}, \mathtt{Gsodh}^b)\\
&+ \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}_{esalt,pi}, \mathtt{Gpi}^b_{esalt}) + \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}_{O^*,pi}, \mathtt{Gpi}^b_{O^*})+\\
&\sum_{\ell=0}^{d-1} \big(\mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}_{\mathrm{es},\ell}, \mathtt{Gxtr}^b_{es,\ell}) + \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}_{\mathrm{hs},\ell}, \mathtt{Gxtr}^b_{hs,\ell})+\\
&\qquad \mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}_{\mathrm{as}}, \mathtt{Gxtr}^b_{as,\ell}) + \sum_{n\in XPR} \big(\mathsf{Adv}(\mathcal{A}_i \to \mathcal{R}_{\mathrm{n},\ell}, \mathtt{Gxpd}^b_{n,\ell})\big)\big),
\end{aligned}
$$



(a) Game $\texttt{Gcore}^{\mathsf{SODH}}$ if $b = 0$ and game $\texttt{Gcore}^{\mathsf{SODH}}$ if $b = 1$. Reduction $\mathcal{R}_{\mathsf{SODH}}$ is marked in grey.

$\underline{\texttt{Rxpd}_{n,\ell}}$

$\underline{\mathsf{XPD}_{n,\ell}(h_1, r, args)}$

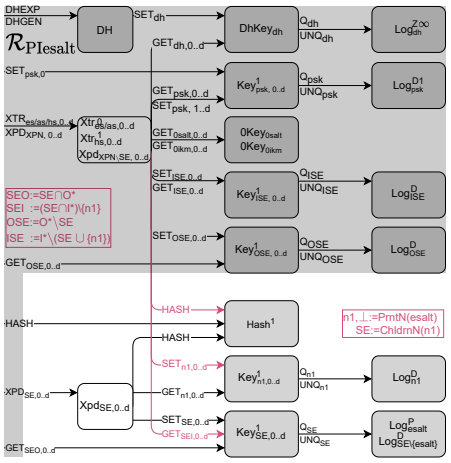
$n_1, \_ \leftarrow \mathsf{PrntN}(n)$
$label \leftarrow \mathsf{Labels}(n, r)$
$h \leftarrow \mathsf{xpd}\langle n, label, h_1, args\rangle$
$(k_1, hon) \leftarrow \mathsf{GET}_{n_1,\ell}(h_1)$
$d \leftarrow \mathsf{HASH}(args)$
$k \leftarrow \mathsf{HASH}_{\mathsf{xpd}}(k_1, (label, d))$
if $n = psk$:
  $h \leftarrow \mathsf{SET}_{n,\ell+1}(h, hon, k)$
else
  $h \leftarrow \mathsf{SET}_{n,\ell}(h, hon, k)$
return h

$\underline{\texttt{Rxtr}^b_{n,\ell}}$

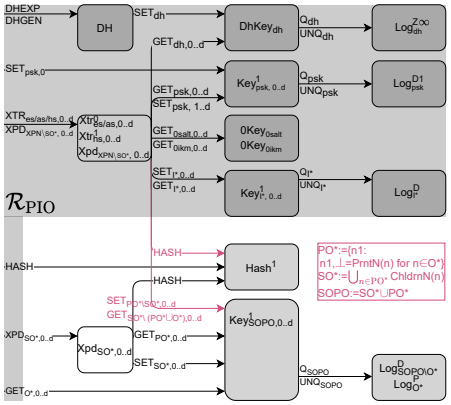
$\underline{\mathsf{XTR}_{n,\ell}(h_1, h_2)}$

$n_1, n_2 \leftarrow \mathsf{PrntN}(n)$
if $\mathsf{alg}(h_1) \neq \bot \wedge \mathsf{alg}(h_2) \neq \bot$:
  assert $\mathsf{alg}(h_1) = \mathsf{alg}(h_2)$
$h \leftarrow \mathsf{xtr}\langle n, h_1, h_2\rangle$
$(k_1, hon_1) \leftarrow \mathsf{GET}_{n_1,\ell}(h_1)$
$(k_2, hon_2) \leftarrow \mathsf{GET}_{n_2,\ell}(h_2)$
$k \leftarrow \mathsf{HASH}_{\mathsf{xtr}}(k_1, k_2)$
$hon \leftarrow hon_1 \vee hon_2$
if $b \wedge hon_2$:
  $k^\star \leftarrow_{\$} \{0,1\}^{\mathsf{len}(k)}$
  $k \leftarrow \mathsf{tag}_{\mathsf{alg}(k)}(k^\star)$
$h \leftarrow \mathsf{SET}_{n,\ell}(h, hon, k)$
return h

(b) $\texttt{Rxtr}_n$ for $n \in \{es, hs, as\}$ and $\texttt{Rxpd}_n$ for $n \in XPN$.

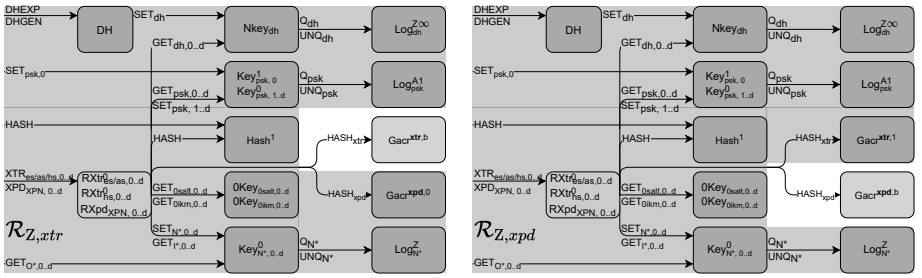


(c) Game $\texttt{Gcore}^{\mathsf{ki}}$ if $P = R$. Reduction $\mathcal{R}_{\mathsf{PIesalt}}$ is marked in grey.



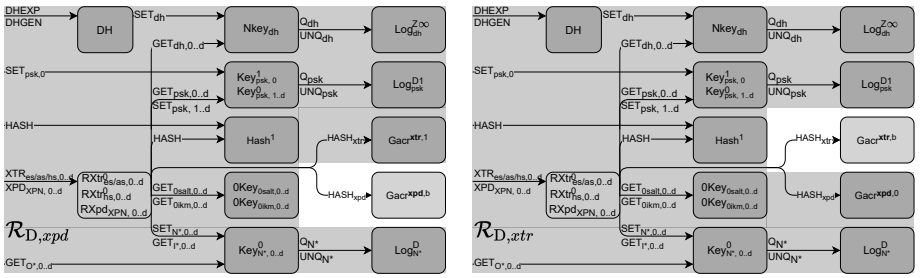
(d) Game $\texttt{Gcore}^{1}(\mathcal{S}^{\mathrm{core}})$ if $P = F$. Reduction $\mathcal{R}_{\mathsf{PIO}}$ is marked in grey.

Fig. 32: Reductions for the lemmas related to Theorem D.1



(a) Game $\mathcal{R}_{Z,\mathtt{xtr}} \rightarrow \mathtt{Garc}^{\mathsf{xtr},b}$, reduction $\mathcal{R}_{Z,\mathtt{xtr}}$ in grey.

(b) Game $\mathcal{R}_{Z,\mathtt{xpd}} \rightarrow \mathtt{Garc}^{\mathsf{xpd},b}$, reduction $\mathcal{R}_{Z,\mathtt{xpd}}$ in grey.



(c) Game $\mathcal{R}_{D,\mathtt{xpd}} \rightarrow \mathtt{Garc}^{\mathsf{xpd},b}$, reduction $\mathcal{R}_{D,\mathtt{xpd}}$ in grey.

(d) Game $\mathcal{R}_{D,\mathtt{xtr}} \rightarrow \mathtt{Garc}^{\mathsf{xtr},b}$, reduction $\mathcal{R}_{D,\mathtt{xtr}}$ in grey.

Fig. 33: Reductions for Lemma D.2

D.2 Proof of Lemma D.3

The high-level idea for the proof of Lemma D.3 is to (a) first idealize collision-resistance for xtr and xpd, then (b) introduce D abort patterns in a perfect equivalence step and then (c) de-idealize collision-resistance of xtr and xpd.

Hybrid argument For each $n \in \{es, hs, as\}$, we decompose $\mathtt{Xtr}_n$ into $\mathtt{Rxtr}_n$ and $\mathtt{Gacr}^{\mathsf{xtr},0}$, and for each $n \in XPR$, we decompose $\mathtt{Xpd}_n$ into $\mathtt{Rxpd}_n$ and $\mathtt{Gacr}^{\mathsf{xpd},0}$, see Fig. 33a (using $b = 0$) for the resulting game $\mathcal{R}_{Z,\mathtt{xtr}} \rightarrow \mathtt{Garc}^{\mathsf{xtr},0}$ and Fig. 32b for the code of $\mathtt{Rxtr}_n$, the difference with $\mathtt{Xtr}_n$ and $\mathtt{Xpd}_n$ is marked in pink.

We establish the following code equivalences:

$$\mathtt{Gcore}^0 \overset{\text{code}}{\equiv} \mathcal{R}_{Z,\mathtt{xtr}} \rightarrow \mathtt{Garc}^{\mathsf{xtr},0} \tag{21}$$

$$\mathcal{R}_{Z,\mathtt{xtr}} \rightarrow \mathtt{Garc}^{\mathsf{xtr},1} \overset{\text{code}}{\equiv} \mathcal{R}_{Z,\mathtt{xtr}} \rightarrow \mathtt{Garc}^{\mathsf{xpd},0} \tag{22}$$

$$\mathcal{R}_{Z,\mathtt{xtr}} \rightarrow \mathtt{Garc}^{\mathsf{xpd},1} \overset{\text{code}}{\equiv} \mathcal{R}_{D,\mathtt{xtr}} \rightarrow \mathtt{Garc}^{\mathsf{xpd},1} \tag{23}$$

$$\mathcal{R}_{D,\mathtt{xtr}} \rightarrow \mathtt{Garc}^{\mathsf{xpd},0} \overset{\text{code}}{\equiv} \mathcal{R}_{D,\mathtt{xtr}} \rightarrow \mathtt{Garc}^{\mathsf{xtr},1} \tag{24}$$

$$\mathcal{R}_{D,\mathtt{xtr}} \rightarrow \mathtt{Garc}^{\mathsf{xtr},0} \overset{\text{code}}{\equiv} \mathtt{Gcore}^D \tag{25}$$

We turn to proving the code equivalences (21)-(25) shortly and now first show that (21)-(25) together imply Lemma D.3. The proof is a standard hybrid argument which we include for completeness. Below, we mark in pink the terms which are 0 due to (22)-(24). For all adversaries $\mathcal{A}$, we have:

$$
\begin{aligned}
& \mathsf{Adv}(\mathcal{A}, \mathtt{Gcore}^{\mathsf{Hash}}, \mathtt{Gcore}^{D}) \\
\overset{(21)+(25)}{=} & \mathsf{Adv}(\mathcal{A}, \mathcal{R}_{Z,\mathsf{xtr}} \to \mathtt{Garc}^{\mathsf{xtr},0}, \mathcal{R}_{D,\mathsf{xtr}} \to \mathtt{Garc}^{\mathsf{xtr},0}) \\
\leq & \mathsf{Adv}(\mathcal{A}, \mathcal{R}_{Z,\mathsf{xtr}} \to \mathtt{Garc}^{\mathsf{xtr},0}, \mathcal{R}_{Z,\mathsf{xtr}} \to \mathtt{Garc}^{\mathsf{xtr},1}) \\
& + \mathsf{Adv}(\mathcal{A}, \mathcal{R}_{Z,\mathsf{xtr}} \to \mathtt{Garc}^{\mathsf{xtr},1}, \mathcal{R}_{Z,\mathsf{xpd}} \to \mathtt{Garc}^{\mathsf{xpd},0}) \\
& + \mathsf{Adv}(\mathcal{A}, \mathcal{R}_{Z,\mathsf{xpd}} \to \mathtt{Garc}^{\mathsf{xpd},0}, \mathcal{R}_{Z,\mathsf{xpd}} \to \mathtt{Garc}^{\mathsf{xpd},1}) \\
& + \mathsf{Adv}(\mathcal{A}, \mathcal{R}_{Z,\mathsf{xpd}} \to \mathtt{Garc}^{\mathsf{xpd},1}, \mathcal{R}_{D,\mathsf{xpd}} \to \mathtt{Garc}^{\mathsf{xpd},1}) \\
& + \mathsf{Adv}(\mathcal{A}, \mathcal{R}_{D,\mathsf{xpd}} \to \mathtt{Garc}^{\mathsf{xpd},1}, \mathcal{R}_{D,\mathsf{xpd}} \to \mathtt{Garc}^{\mathsf{xpd},0}) \\
& + \mathsf{Adv}(\mathcal{A}, \mathcal{R}_{D,\mathsf{xpd}} \to \mathtt{Garc}^{\mathsf{xpd},0}, \mathcal{R}_{D,\mathsf{xtr}} \to \mathtt{Garc}^{\mathsf{xtr},1}) \\
& + \mathsf{Adv}(\mathcal{A}, \mathcal{R}_{D,\mathsf{xtr}} \to \mathtt{Garc}^{\mathsf{xtr},1}, \mathcal{R}_{D,\mathsf{xtr}} \to \mathtt{Garc}^{\mathsf{xtr},0}) \\
\leq & \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{\mathrm{Z},\mathsf{xtr}}, \mathtt{Gacr}^{\mathsf{xtr},b}) + \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{\mathrm{Z},\mathsf{xpd}}, \mathtt{Gacr}^{\mathsf{xpd},b}) \\
& + \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{\mathrm{D},\mathsf{xtr}}, \mathtt{Gacr}^{\mathsf{xtr},b}) + \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{\mathrm{D},\mathsf{xpd}}, \mathtt{Gacr}^{\mathsf{xpd},b}),
\end{aligned}
$$

It remains to prove (21)-(25). $\mathtt{Gcore}^{\mathsf{Hash}}$ is functionally equivalent to $\mathcal{R}_{Z,\mathsf{xtr}} \to \mathtt{Garc}^{\mathsf{xtr},0}$ by inlining the code of the $\mathsf{HASH}_{\mathsf{xtr}}$ and $\mathsf{Hash}_{\mathsf{xpd}}$ oracles of $\mathtt{Gacr}^{\mathsf{xtr},0}$ and $\mathtt{Gacr}^{\mathsf{xpd},0}$. After inlining, the only difference is the line which asserts that the input is in the domain of the function to be analyzed. We here observe that indeed, the inputs are correctly formatted, i.e., $(k_1, k_2) \in \mathrm{dom}(\mathsf{xtr})$, $k_1, (\mathit{label}, d) \in \mathrm{dom}(\mathsf{xpd})$. Therefore, (21) holds. Analogously, (25) holds. Equations (22) and (24) follow by graph pattern matching in Figure 33. The only code equivalence which remains to prove is (23) which we formulate in the following claim:

Claim D.7.1

$$\mathcal{R}_{Z,\mathsf{xtr}} \to \mathtt{Garc}^{\mathsf{xpd},1} \overset{code}{\equiv} \mathcal{R}_{D,\mathsf{xtr}} \to \mathtt{Garc}^{\mathsf{xpd},1}$$

Aborts of the hash-functions cover aborts in the key packages We now prove Claim D.7.1. To show that $\mathcal{R}_{Z,\mathsf{xtr}} \to \mathtt{Garc}^{\mathsf{xpd},1}$ is code-equivalent to $\mathcal{R}_{D,\mathsf{xtr}} \to \mathtt{Garc}^{\mathsf{xpd},1}$, we prove that none of the D-patterns in the $\mathtt{Log}_n$ packages ever trigger an abort. Towards this goal, for each $n \in \mathcal{E} := N \setminus \{\mathit{dh}, \mathit{0salt}, \mathit{0ikm}\}$, we define an event E_n in $\mathcal{R}_{D,\mathsf{xtr}} \to \mathtt{Garc}^{\mathsf{xpd},1}$, namely, the event that *if* the $\mathtt{Log}_n$ *had* a D pattern, then it would abort. The event E_n can be evaluated as a predicate on the current $\mathtt{Log}_n$ state and the input to the UNQ_n query. We define the events as *monotonous*, i.e., if event E_n only has a starting time but no end time.

We note the following properties about the events E_n, $n \in \mathcal{E}$:

(0) For all $n \in \mathcal{E}$: E_n does not holds at time 0 in $\mathcal{R}_{Z,\mathsf{xtr}} \to \mathtt{Garc}^{\mathsf{xpd},1}$.
(1) If E_n happens for the first time in time t, then no other event $E_{n'}$ with $n' \neq n$ happens for the first time in time t.

(2) If E_n holds at time t_1, then there must be a time $t_2 \leq t_1$ and $n' \in \mathcal{E}$ with $n' \neq n$ such that $E_{n'}$ holds at time t_2.

(0) holds by definition of the events. (1) holds because the events are all defined on disjoint sets of variables since they are defined on disjoint packages. The interesting argument is (2) which we prove in Claim D.7.2. Given (0), (1) and (2), by Lemma D.9, the events E_n are what we call *co-dependent* events, i.e., one of them always requires another one to have happened before, and thus, none of the events E_n ever occurs in the game. Thus, we can insert the line **if** $hon = hon' = 0$: **throw** *abort* to the $\mathtt{Log}_n$ code of all $n\mathcal{E}$ without changing the behaviour of the game, i.e., we can switch all patterns from Z to D. This concludes the proof of Claim D.7.1, lest the proofs of Lemma D.9 and of (2). We now first prove (2).

Claim D.7.2 *If E_n holds at time t_1, then there must be a time $t_2 \leq t_1$ and $n' \in \mathcal{E}$ with $n' \neq n$ such that such that $E_{n'}$ holds at time t_2.*

Proof. We idealized `xtr` and `xpd` w.r.t. collision-resistance, key derivations are *injective* operations. We call a *collision* a situation where the same key value is stored under two different handles in some key table $K_{n,\ell}$ with $n \in \mathcal{E}$. Given a key's handle, we can uniquely identify the input key value and its arguments, and if the mapping from handles to keys was injective so far and there is no collision on the `xtr` or `xpd` operation, then the mapping continues to be injective. Dishonest keys are always derived exclusively from dishonest values and due to the ∞ mapping pattern, dishonest Diffie-Hellman keys are unique. Thus, for all derived key names $n \in \mathcal{E}$ with $n \neq psk$, Claim D.7.2 follows.

The most interesting case is $n = \text{psk}$, since dishonest application PSKs are adversarially chosen and not derived values. By the previous argument, we already know that, amongst dishonest *resumption* PSKs, no collisions occur. Additionally, due to the $A1$ pattern in $\mathtt{KEY}_{\text{psk},0..d}$, we already know that amongst dishonest *application* PSKs, no collisions occur. It remains to argue about collisions between dishonest resumption PSKs and dishonest *application* PSKs. Note that due to the mapping for $\mathtt{KEY}_{\text{psk},0..d}$, the first such collision (per key value) is removed such that $P(r, hon, r', hon')$ is never executed. Additionally, note that a 3-way collision already implies that there must be a collision either amongst two dishonest application PSKs or amongst two dishonest resumption PSKs, which we already know cannot occur, which concludes the proof of Claim D.7.2.

D.3 Co-Dependence Lemma

In this appendix, we show that if two non-simultaneous events circularly depend on each other and if they do not hold at time 0, then they cannot occur. That is, to show that two events E_1 and E_2 do not occur, it suffices to show that (a) neither E_1 nor E_2 hold at time 0 and that (b) if one of the events occurs, the other must have already occurred in time strictly before. The proof of this statement proceeds via induction over time. We call such circularly depending events *co-dependent*.

Induction over time

We abstract time by the number of queries t made by the adversary.

Definition D.8 (Co-dependent Events). *Let G be a game, let N be a set, and for $n \in S$, let E_n be an event on G. We call E_n, $n \in N$* co-dependent *on G if they satisfy the following properties:*

(0) For all $n \in N$: E_n does not holds at time 0 on G.
(1) If E_n happens for the first time in time t, then no other event $E_{n'}$ with $n' \neq n$ happens for the first time in time t.
(2) If E_n holds at time t_1, then there must be a time $t_2 \leq t_1$ and $n' \in S$ with $n' \neq n$ such that such that $E_{n'}$ holds at time t_2.

Lemma D.9 (Co-Dependance). *Let G be a game, let N be a set, and for $n \in N$, let E_n be an event on G. If the events E_n, $n \in S$ are* co-dependent *on G, then for all adversaries $\mathcal{A}$ and for all $n \in N$, we have*

$$\Pr[E_n \text{ in } \mathcal{A} \to G] = 0.$$

Proof. By induction over the time. By (0), neither of the events E_n, $n \in S$, holds at time 0. Assume towards contradiction, that there is a time point t and an event E_n such that E_n holds at time t. Then, there is a smallest $t_1 \leq t$ such that there exists some $n' \in N$ such that $E_{n'}$ holds at time t_1. By (2), there is a time point $t_0 \leq t_1$ and an event $E_{n''}$ with $n'' \neq n'$ such that $E_{n''}$ holds at time t_0. By (1), it must be that $t_0 < t_1$ in contradiction to t_1 being the smallest time point with an event holding at the time. Thus, such a smallest time point cannot exist.

D.4 Proof of Lemma D.4 (Resalt)

In order to prove Lemma D.4, given an adversary $\mathcal{A}$, we define an adversary $\mathcal{A}_0$ and an adversary $\mathcal{A}_1$ such that one of them has a bigger advantage than $\mathcal{A}$ and then, Lemma D.4 follows. Namely, for $i \in \{0, 1\}$, we define $\mathcal{A}_i$ as running $\mathcal{A}$ internally and observing whether the game aborts with *win*. If so, then $\mathcal{A}_i$ returns i. Else, $\mathcal{A}_i$ runs $\mathcal{A}$ until $\mathcal{A}$ terminates and returns whatever $\mathcal{A}$ returns. Note that by construction of our simulator $\mathcal{S}^{\text{core}}$, the game $\mathtt{Gcore}^1(\mathcal{S}^{\text{core}})$ never aborts with *win* and thus, whenever the symbol *win* appears, the adversary must be interacting with $\mathtt{Gcore}^{\mathsf{Resalt}}$ and not $\mathtt{Gcore}^1(\mathcal{S}^{\text{core}})$ so that the distinguishing advantage increases by returning the *right* fixed bit i in this case. We now perform the probability analysis via up-to-bad reasoning for completeness.

We denote by $\mathcal{E}_R$ the event that a game returns *win*. Conditioning on event $\mathcal{E}_R$ not occurring, $\mathcal{A}_i$ and $\mathcal{A}$ behave identically:

$$\alpha := \Pr\left[\mathcal{A} \to \mathtt{Gcore}^1(\mathcal{S}^{\text{core}}) = 1\right] = \Pr\left[\mathcal{A}_i \to \mathtt{Gcore}^1(\mathcal{S}^{\text{core}}) = 1\right]$$
$$\beta := \Pr\left[\mathcal{A} \to \mathtt{Gcore}^D = 1 | \neg\mathcal{E}_R\right] = \Pr\left[\mathcal{A}_i \to \mathtt{Gcore}^{\mathsf{Resalt}} = 1 | \neg\mathcal{E}_R\right]$$

Moreover, in $\texttt{Gcore}^1(\mathcal{S}^{\text{core}})$, the event $\mathcal{E}_R$ does not occur. This means that for $i \in \{0,1\}$

$$\begin{aligned}\alpha &= \Pr\left[\mathcal{A} \to \texttt{Gcore}^1(\mathcal{S}^{\text{core}}) = 1\right] = \Pr\left[\mathcal{A}_i \to \texttt{Gcore}^1(\mathcal{S}^{\text{core}}) = 1\right]\\ \beta &= \Pr\left[\mathcal{A} \to \texttt{Gcore}^D = 1 | \neg\mathcal{E}_R\right] = \Pr\left[\mathcal{A}_i \to \texttt{Gcore}^{\mathsf{Resalt}} = 1 | \neg\mathcal{E}_R\right]\\ &\Pr\left[\mathcal{A}_i \to \texttt{Gcore}^{\mathsf{Resalt}} = 1 | \mathcal{E}_R\right] = i\end{aligned}$$

We can now prove Lemma D.4 as follows:

$$\mathsf{Adv}(\mathcal{A}, \texttt{Gcore}^D, \texttt{Gcore}^1(\mathcal{S}^{\text{core}})) \stackrel{\text{def}}{=} \tag{26}$$

$$\left|\Pr\left[\mathcal{A} \to \texttt{Gcore}^D = 1\right] - \Pr\left[\mathcal{A} \to \texttt{Gcore}^1(\mathcal{S}^{\text{core}}) = 1\right]\right| \tag{27}$$

$$= \left|\Pr[\mathcal{E}_R] \cdot \Pr\left[\mathcal{A} \to \texttt{Gcore}^D = 1 \;\middle|\; \mathcal{E}_R\right] + \Pr[\neg\mathcal{E}_R] \cdot \beta - \alpha\right|$$

If $\Pr[\mathcal{E}_R] \cdot \Pr\left[\mathcal{A} \to \texttt{Gcore}^D = 1 \;\middle|\; \mathcal{E}_R\right] + \Pr[\neg\mathcal{E}_R] \cdot \beta - \alpha \leq 0$, we upper bound (27) by $\Pr[\mathcal{E}_R] + \Pr[\neg\mathcal{E}_R] \cdot \beta - \alpha$. If $\alpha - \Pr[\mathcal{E}_R] \cdot \Pr\left[\mathcal{A} \to \texttt{Gcore}^D = 1 \;\middle|\; \mathcal{E}_R\right] - \Pr[\neg\mathcal{E}_R] \cdot \beta \leq 0$, then (27) is upper bounded by $\alpha - \Pr[\mathcal{E}_R] \cdot \Pr\left[\mathcal{A} \to \texttt{Gcore}^D = 1 \;\middle|\; \mathcal{E}_R\right]$ and thus, (27) is equal or smaller than

$$\begin{aligned}&\leq \max\begin{pmatrix} | & \Pr[\neg\mathcal{E}_R] \cdot \beta - \alpha |, \\ |\Pr[\mathcal{E}_R] + \Pr[\neg\mathcal{E}_R] \cdot \beta - \alpha| & \end{pmatrix}\\ &= \max_{i \in \{0,1\}} \left|\Pr[\mathcal{E}_R] \Pr\left[\mathcal{A}_i \to \texttt{Gcore}^{\mathsf{Resalt}} = 1 \;\middle|\; \mathcal{E}_R\right] + \Pr[\neg\mathcal{E}_R] \cdot \beta - \alpha\right|\\ &= \max_{i \in \{0,1\}} \left(\left|\Pr\left[\mathcal{A}_i \to \texttt{Gcore}^{\mathsf{Resalt}} = 1\right] - \Pr\left[\mathcal{A}_i \to \texttt{Gcore}^1(\mathcal{S}^{\text{core}}) = 1\right]\right|\right)\\ &= \mathsf{Adv}(\mathcal{A}, \texttt{Gcore}^{\mathsf{Resalt}}, \texttt{Gcore}^1(\mathcal{S}^{\text{core}})),\end{aligned}$$

which concludes the proof of Lemma D.4.

D.5 Proof of Lemma D.5

We can separate out the graph of the $\texttt{Gsodh}^0$ game from $\texttt{Gcore}^{\mathsf{Resalt}}$, and call the remaining part of the graph the reduction $\mathcal{R}_{\text{sodh}}$, see Fig. 32a. We then have that

$$\texttt{Gcore}^{\mathsf{Resalt}} \stackrel{\text{code}}{\equiv} \mathcal{R}_{\text{sodh}} \to \texttt{Gsodh}^0.$$

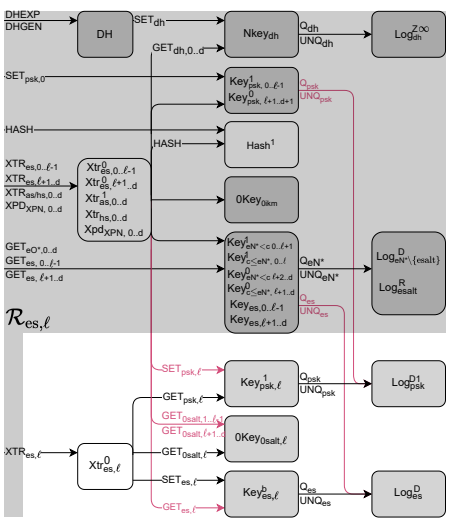
Similarly, we can separate out the graph of the $\texttt{Gsodh}^1$ game from $\texttt{Gcore}^{\mathsf{SODH}}$, using the same remaining graph $\mathcal{R}_{\text{sodh}}$, see Fig. 32a, and we obtain

$$\texttt{Gcore}^{\mathsf{SODH}} \stackrel{\text{code}}{\equiv} \mathcal{R}_{\text{sodh}} \to \texttt{Gsodh}^1.$$

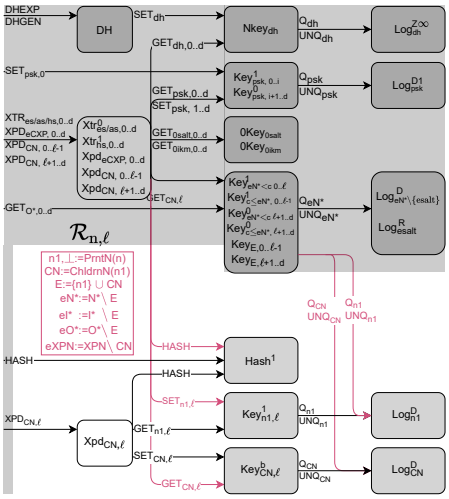
Putting these two equations together, we obtain

$$\begin{aligned}&\mathsf{Adv}(\mathcal{A}, \texttt{Gcore}^{\mathsf{Hash}}, \texttt{Gcore}^{\mathsf{SODH}})\\ &= \left|\Pr\left[\mathcal{A} \to \texttt{Gcore}^{\mathsf{Hash}}\right] - \Pr\left[\mathcal{A} \to \texttt{Gcore}^{\mathsf{SODH}}\right]\right|\\ &= \left|\Pr\left[\mathcal{A} \to (\mathcal{R}_{\text{sodh}} \to \texttt{Gsodh}^0\right]) - \Pr\left[\mathcal{A} \to (\mathcal{R}_{\text{sodh}} \to \texttt{Gsodh}^1\right])\right|\\ &= \left|\Pr\left[(\mathcal{A} \to \mathcal{R}_{\text{sodh}}) \to \texttt{Gsodh}^0\right] - \Pr\left[(\mathcal{A} \to \mathcal{R}_{\text{sodh}}) \to \texttt{Gsodh}^1\right]\right|\\ &= \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{\text{sodh}}, \texttt{Gsodh}^b)\end{aligned}$$

This concludes the proof of Lemma D.5.



(a) Reduction $\mathcal{R}_{n,\ell}$ is marked in grey for $n \in XPN \setminus \{psk, esalt\}$. Reductions $\mathcal{R}_{psk,\ell}$ and $\mathcal{R}_{esalt,\ell}$ are defined analogously.



(b) Reduction $\mathcal{R}_{n,\ell}$ is marked in grey for $n \in XPN \setminus \{psk, esalt\}$, where $\mathcal{R}_{n,\ell}$ is defined in Fig. 34b. The reductions for $n \in \{psk, esalt\}$ are defined analogously.

Fig. 34: Reductions for the proof of Lemma D.6.

D.6 Idealization Order and additional Notation

Before turning to the proof of Lemma D.6 we introduce the concept of *idealization order* as well as notation for sets (used throughout the paper and summarized in Table 1).

N	The set of all (key) names
N^*	$N \setminus \{psk, dh\}$
I^*	The set of internal keys $\{n \in N^* \mid \mathsf{ChldrnOp}(n) = out\}$
O^*:	The set of output keys $\{n \in N^* \mid \mathsf{ChldrnOp}(n) = out\}$
O:	$O^* \cup \{psk\}$
S:	The set of separation points (Definition 4.4)
XPN:	The set of expand names $\{n \in N : \mathsf{PrntOp}(n) = \mathsf{xpd}\}$
XPR:	The set of representatives (Section 4.7)

Table 1: Notation

Additional Notation The function PrntN of the key schedule syntax (Definition 4.1) induce a parent operation function PrntOp, a children name function ChldrnN, a sibling name function SblngN and a children operation function

ChldrnOp. By slight abuse of notation, we use $\mathsf{PrntOp}(\mathsf{ChldrnN}(n))$ to denote the (identical) parent operation of all children of n.

$$
\begin{aligned}
&\mathsf{ChldrnN}: && N \to \{0,1\}^N \\
& && n \mapsto \left\{ n' \in N \;\middle|\; \begin{matrix} \mathsf{PrntN}(n') = n, \_ \vee \\ \mathsf{PrntN}(n') = \_, n \end{matrix} \right\} \\
&\mathsf{SblngN}: && N \to \{0,1\}^N \\
& && n \mapsto \{ n' \in N \mid \mathsf{PrntN}(n) = \mathsf{PrntN}(n') \} \\
&\mathsf{PrntOp}: && N \to \{ \mathit{base}, \mathit{xtr}, \mathit{xpd} \} \\
& && n \mapsto \begin{cases} \mathit{base} & \text{if } \mathsf{PrntN}(n) = (\bot, \bot) \\ \mathit{xpd} & \text{if } \mathsf{PrntN}(n) = (\_, \bot) \\ \mathit{xtr} & \text{Otherwise} \end{cases} \\
&\mathsf{ChldrnOp}: && N \to \{ \mathit{xt}, \mathit{xp}, \mathit{out} \} \\
& && n \mapsto \begin{cases} \mathit{out} & \text{if } \mathsf{ChldrnN}(n) = \emptyset \\ \mathsf{PrntOp}(\mathsf{ChldrnN}(n)) & \text{otherwise} \end{cases}
\end{aligned}
$$

Idealization Order For the inner hybrid argument over the graph structure of the key schedule (Lemma D.6), we need to follow an *idealization order*—intuitively we reduce to the assumptions once the `Key` packages for the parent names have been idealized which allows us to idealize the `Key` packages of the output keys as well.

While this is similar to the order in which the key schedule is executed, there is an significant difference. The key schedule computes one *output key* per step while the idealization proceeds with one *operation* or input key. While generating one key at a time is more natural in the functional interface of the key schedule (and sometimes necessary as transcripts may depend on keys siblings) the xpd assumption crucially relies on domain separation between the siblings and therefore needs to consider them all at the same time.

$\mathsf{IdealizationOrder}(ks, <)$

1 : $\mathsf{io}[1] := \{\mathit{psk}, \mathit{0salt}, \mathit{dh}, \mathit{0ikm}\}$
2 : $c \leftarrow 1$
3 : **while** $\mathsf{io}[c] \subsetneq N$
4 : $\quad n^c \leftarrow \min\{n \in N \setminus \mathsf{io}[c] :$
5 : $\qquad (n_1, n_2) \leftarrow \mathsf{PrntN}(n) \wedge$
6 : $\qquad \{n_1, n_2\} \subseteq \mathsf{io}[c]_\bot \wedge$
7 : $\qquad \mathsf{SblngN}(n) \not\subseteq \mathsf{io}[c]\}$
8 : $\quad \mathsf{io}[c+1] \leftarrow \mathsf{io}[c] \cup \mathsf{SblngN}(n^c)$
9 : $\quad c \leftarrow c + 1$
10 : $m \leftarrow c$
11 : **return** $\mathsf{io}, m, (n^1, \ldots, n^{m-1})$

Fig. 35: Idealization Order

Definition D.10 (Idealization Order). *Let $ks = (N, \mathsf{Label}, \mathsf{PrntN})$ be a TLS-like key schedule and $m \in \mathbb{N}$. An* Idealization Order io *for ks is a total order on m subsets of N together with a sequence of names $n^1, \ldots, n^{m-1}$ such that*

- $\mathsf{io}[1] = \{\mathit{psk}, \mathit{0salt}, \mathit{dh}, \mathit{0ikm}\}$

- $\mathsf{io}[1] \subset \cdots \subset \mathsf{io}[m] = N$
- $\forall 1 \leq c < m : n^c \in \mathsf{io}[c], \mathsf{io}[c+1] = \mathsf{io}[c] \cup \mathsf{SblngN}(n^c)$

For $1 \leq c \leq m$ we denote by $N < c$ the set of names before c, $N \cap \mathsf{io}[c]$ and by $c \leq N$ the set of names after c, $N \setminus \mathsf{io}[c]$

Claim D.10.1 (Idealization Order) *Every TLS-like key schedule has an idealization order.*

Proof. Let $<$ be a total order on N. Note that min computes the minimum with respect to $<$. The assignment in line 4 is well-defined: Let n be in $N \setminus \mathsf{io}[c]$. Since n is reachable from *psk* (Definition 4.2), there is a path from *psk* to n. As there is a first element on this path not in $\mathsf{io}[c]$, the set $\{n \in N \setminus \mathsf{io}[c] : (n_1, n_2) \leftarrow \mathsf{PrntN}(n) \ \wedge \ \{n_1, n_2\} \subseteq \mathsf{io}[c]_\perp \ \wedge \ \mathsf{SblngN}(n) \not\subseteq \mathsf{io}[c]\}$ is non-empty. As $\mathsf{io}[c] \subsetneq \mathsf{io}[c+1]$ and N is finite, the algorithm terminates and $\mathsf{io}[m] = N$

D.7 Proof of Lemma D.6

The proof of Lemma D.6 relies on the idealization order (Definition D.10), which is guaranteed to exist by Claim D.10.1.

Lemma D.11 (XTR1). *For $n^c = psk$, we have that*

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gks}^{hyb:\ell,c}, \mathtt{Gks}^{hyb:\ell,c+1}) \leq \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{es,\ell}, \mathtt{Gxtr}^b_{es,\ell}),$$

where $\mathcal{R}_{es,\ell}$ is defined in Fig. 34a.

Lemma D.12 (XTR2). *For $n^c = esalt$*

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gks}^{hyb:\ell,c}, \mathtt{Gks}^{hyb:\ell,c+1}) \leq \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{hs,\ell}, \mathtt{Gxtr}^b_{hs,\ell}),$$

where $\mathcal{R}_{hs,\ell}$ is defined analogously to reduction $\mathcal{R}_{es,\ell}$ in Fig. 34a.

Lemma D.13 (XTR3). *For $n^c = hsalt$, we have that*

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gks}^{hyb:\ell,c}, \mathtt{Gks}^{hyb:\ell,c+1}) \leq \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{as,\ell}, \mathtt{Gxtr}^b_{as,\ell}),$$

where $\mathcal{R}_{as,\ell}$ is defined analogously to reduction $\mathcal{R}_{es,\ell}$ in Fig. 34a.

Lemma D.14 (XPD). *For $\mathsf{ChldrnOp}(n^c) = \mathsf{xpd}$, we have that*

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gks}^{hyb:\ell,c}, \mathtt{Gks}^{hyb:\ell,c+1}) \leq \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{n,\ell}, \mathtt{Gxpd}^b_{n,\ell}),$$

where Game $\mathtt{Gxpd}^b_{n,\ell}$ for $b \in \{0,1\}$ is defined in Fig. 24a for $n \notin \{psk, esalt\}$, in Fig. 24b for $n = esalt$ and in Fig. 24c for $n = psk$. For $n \notin \{esalt, esalt\}$, where $\mathcal{R}_{n,\ell}$ is defined in Fig. 34b. The reductions for $n \in \{psk, esalt\}$ are defined analogously.

From Lemma D.11, Lemma D.12, Lemma D.13, and Lemma D.14 we obtain Lemma D.6 by observing that

$$\mathtt{Gks}^{\mathrm{hyb}:\ell} = \mathtt{Gks}^{\mathrm{hyb}:\ell,1} \text{ and } \mathtt{Gks}^{\mathrm{hyb}:\ell+1} = \mathtt{Gks}^{\mathrm{hyb}:\ell,m}$$

and a standard hybrid argument, which we include for completeness:

$$\begin{aligned}
&\mathsf{Adv}(\mathcal{A}, \mathtt{Gks}^{\mathrm{hyb}:\ell}, \mathtt{Gks}^{\mathrm{hyb}:\ell+1}) \\
=&\mathsf{Adv}(\mathcal{A}, \mathtt{Gks}^{\mathrm{hyb}:\ell,1}, \mathtt{Gks}^{\mathrm{hyb}:\ell,m}) \\
\leq &\sum_{c=1}^{m-1} \mathsf{Adv}(\mathcal{A}, \mathtt{Gks}^{\mathrm{hyb}:\ell,c}, \mathtt{Gks}^{\mathrm{hyb}:\ell,c+1}) \\
=&\mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{\mathrm{es},\ell}, \mathtt{Gxtr}^b_{es,\ell}) + \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{\mathrm{hs},\ell}, \mathtt{Gxtr}^b_{hs,\ell}) \\
&+ \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{\mathrm{as},\ell}, \mathtt{Gxtr}^b_{as,\ell}) + \sum_{n \in XPR} \big(\mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{\mathrm{n},\ell}, \mathtt{Gxpd}^b_{n,\ell})\big),
\end{aligned}$$

Each of the lemmas are derived using the following pattern. We have a big game $\mathtt{G}^b_{\mathrm{big}}$ which we split into a reduction $\mathcal{R}$ and a smaller game $\mathtt{G}^b_{\mathrm{small}}$, i.e., $\mathtt{G}^b_{\mathrm{big}} \overset{\mathrm{code}}{\equiv} \mathcal{R} \to \mathtt{G}^b_{\mathrm{small}}$, and then, we use the following equations:

$$\begin{aligned}
&\mathsf{Adv}(\mathcal{A}, \mathtt{G}^0_{big}, \mathtt{G}^1_{big}) \\
=&\left|\Pr\left[1 = \mathcal{A} \to \mathtt{G}^0_{big}\right] - \Pr\left[1 = \mathcal{A} \to \mathtt{R} \to \mathtt{G}^1_{big}\right]\right| \\
=&\left|\Pr\left[1 = \mathcal{A} \to (\mathtt{R} \to \mathtt{G}^0_{small})\right] - \Pr\left[1 = \mathcal{A} \to (\mathtt{R} \to \mathtt{G}^1_{small})\right]\right| \\
=&\left|\Pr\left[1 = (\mathcal{A} \to \mathtt{R}) \to \mathtt{G}^0_{small})\right] - \Pr\left[1 = (\mathcal{A} \to \mathtt{R}) \to \mathtt{G}^1_{small}\right]\right| \\
=&\mathsf{Adv}(\mathcal{A} \to \mathtt{R}, \mathtt{G}^0_{small}, \mathtt{G}^1_{small})
\end{aligned}$$

E Reductions to Standard Assumptions

In this appendix, we relate the compositional assumptions we introduced in Appendix A to the standard, monolithic assumptions we defined in Section 3. The appendix proceeds in the order in which the assumptions are used in the proof of Theorem 4.5. Appendix E.1 relates the modular $\texttt{Gsodh}^b$ game (Figure 24g) to the monolithic $\texttt{Gsodh2}^{grp,b}$ assumption (Figure 9). Appendix E.2 introduces a parametrized generic game $\texttt{G-M-Pr-io}^{in,\mathsf{f},b,coll_{\mathrm{in}},coll_{\mathrm{Out}}}_{Out}$ which is useful for the proofs of the pseudorandomness and pre-image resistance assumptions. Lemma E.5 states how indistinguishability of $\texttt{G-M-Pr-io}^{in,\mathsf{f},b,coll_{\mathrm{in}},coll_{\mathrm{Out}}}_{Out}$ with $b = 0$ and $b = 1$ relates to the monolithic (non-agile) pseudorandomness game $\texttt{Gpr}^{\mathsf{f}\text{-}alg,b}$ (Figure 8). The proof of this relation is of independent interest and thus carried out in a companion paper [29]. Using Lemma E.5, Appendix E.3 upper bounds the advantage of an adversary $\mathcal{A}$ playing against the pseudorandomness games $\texttt{Gxtr1}_{es,\ell}$, $\texttt{Gxtr2}_{hs,\ell}$, $\texttt{Gxtr3}_{as,\ell}$, $\texttt{Gxpd}_{n,\ell}$ by the advantage of an adversary $\mathcal{A} \to \mathcal{R}$ against the monolithic pseudorandomness game $\texttt{Gpr}^{\mathsf{f}\text{-}alg,b}$ (for a suitable reduction $\mathcal{R}$). Finally, Appendix E.4 upper bounds the advantage of an adversary playing against the pre-image resistance games $\texttt{Gpi}^{P}_{O^*}$ and $\texttt{Gpi}^{P}_{esalt}$ by the advantage of an adversary $\mathcal{A} \to \mathcal{R}$ against the monolithic pseudorandomness game $\texttt{Gpr}^{\mathsf{f}\text{-}alg,b}$ (for a suitable reduction $\mathcal{R}$) as well as collision-resistance of $\texttt{xpd}$. This reduction also relies on Lemma E.5.

E.1 Composed SODH Security to Monolithic SODH Assumption

In this section, we upper bound the advantage of an adversary $\mathcal{A}$ against the game $\texttt{Gsodh}^b$ (see Fig. 24g) by the advantage of an adversary $\mathcal{A} \to \mathcal{R}^{grp}_{\mathrm{sod}h}$ against the monolithic game $\texttt{Gsodh2}^{b,grp}$ (Fig. 9, Section 3.3). Since $\texttt{Gsodh2}^{b,grp}$ considers a fixed group *grp* and $\texttt{Gsodh}^b$ is agile in the group, the proof contains a hybrid argument leading to the upper bound in Lemma E.1 which sums over all groups. In turn, there is no hybrid argument summing over all hash-algorithms, since $\texttt{Gsodh2}^{grp,b}$ is agile in the set of hash-algorithms—the same Diffie-Hellman secret can be used with several hash-functions. The proof of Lemma E.1 additionally invokes collision-resistance of one of the hash-functions supported by $\texttt{Gsodh2}^{b,grp}$. Note that we only rely on the *non-agile* collision-resistance of a single hash-function here. We now restate Lemma E.1 for convenience.

Lemma E.1 (Salted-ODH Advantage). *For all $\mathcal{A}$, it holds that*

$$\mathsf{Adv}(\mathcal{A}, \texttt{Gsodh}^0, \texttt{Gsodh}^1) \leq 2 \cdot \min_{alg \in \mathcal{H}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{alg}_{sodh\text{-}cr}, \texttt{Gcr}^{\mathsf{hash}\text{-}alg,b}) + \sum_{grp \in \mathcal{G}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{grp}_{sodh}, \texttt{Gsodh2}^{grp,b}),$$

where $\mathcal{R}^{alg}_{sodh\text{-}cr} := \mathcal{R}_{sodh4} \to \mathcal{R}^{alg}_{sodh2\text{-}cr,alg}$ and $\mathcal{R}^{grp}_{sodh} := \mathcal{R}_{sodh4} \to \mathcal{R}_{sodh2} \to \mathcal{R}_{grp}$. We define the reductions $\mathcal{R}_{sodh4}$ and $\mathcal{R}_{grp}$ in Lemma E.2 and Lemma E.4, respectively, and we describe $\mathcal{R}^{alg}_{sodh2\text{-}cr,alg}$ and $\mathcal{R}_{sodh2}$ in Lemma E.3.

The proof of Lemma E.1 will directly follow from Lemma E.2, Lemma E.3 and Lemma E.4, which we state now. Afterwards, we show that the three lemmas, together with Lemma A.2 imply Lemma E.1 and then prove each of the three lemmas in turn.

Lemma E.2. *For all adversaries* $\mathcal{A}$

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gsodh}^b) = \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{sodh4}, \mathtt{Gagsodh4}^b),$$

where game $\mathtt{Gagsodh4}^b$ *and reduction* $\mathcal{R}_{sodh4}$ *are defined in Fig. 36a.*

Lemma E.3. *For all adversaries* $\mathcal{A}$

$$\begin{aligned}&\mathsf{Adv}(\mathcal{A}, \mathtt{Gagsodh4}^b)\\ \leq&\mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{sodh2}, \mathtt{Gagsodh2}^b) + 2 \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{alg}_{sodh2\text{-}cr,alg}, \mathtt{Gacr}^{\mathsf{xtr},b}),\end{aligned}$$

where $\mathtt{Gagsodh2}^b$ *and* $\mathcal{R}_{sodh2}$ *are defined in Fig. 40, and* $\mathcal{R}^{alg}_{sodh2\text{-}cr,alg}$ *is described in the proof of Lemma E.3.*

Lemma E.4. *For all adversaries* $\mathcal{A}$

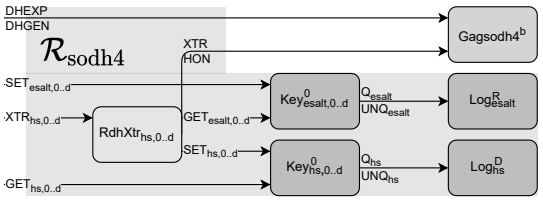
$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gagsodh2}^b) \leq \sum_{grp \in \mathcal{G}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{grp}, \mathtt{Gsodh2}^{grp,b}),$$

where $\mathtt{Gsodh2}^{grp,b}$ *is defined in Fig. 9, and* $\mathcal{R}_{grp}$ *is defined in Fig. 41.*

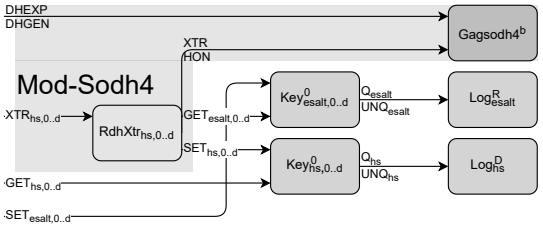
Proof of Lemma E.1 Let $\mathcal{A}$ be an adversary. We derive Lemma E.1 as follows

$$\begin{aligned}
&\mathsf{Adv}(\mathcal{A}, \mathtt{Gsodh}^b)\\
\overset{\text{L. E.2}}{=}\ &\mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{\mathrm{sodh4}}, \mathtt{Gagsodh4}^b)\\
\overset{\text{L. E.3}}{=}\ &\mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{\mathrm{sodh4}} \to \mathcal{R}_{\mathrm{sodh2}}, \mathtt{Gagsodh2}^b) + \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\mathrm{cr}}_{\mathrm{sodh2}}, \mathtt{Gacr}^{\mathsf{xtr},b})\\
\overset{\text{L. E.4}}{=}\ &\sum_{grp \in \mathcal{G}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{\mathrm{sodh4}} \to \mathcal{R}_{\mathrm{sodh2}} \to \mathcal{R}_{grp}, \mathtt{Gsodh2}^b)\\
&+ \min_{alg \in \mathcal{H}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{alg}_{sodh2-cr}, \mathtt{Gcr}^{\mathsf{xtr}\text{-}alg,b}).
\end{aligned}$$

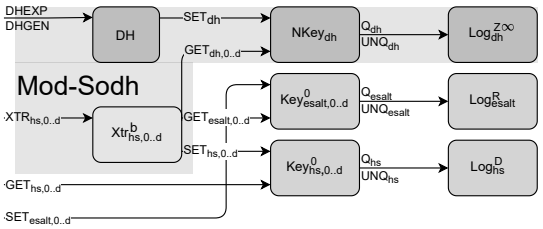
Thus, it remains to prove Lemma E.2, Lemma E.3 and Lemma E.4 each in turn.



(a) Game $\texttt{Gagsodh4}^b$, reduction $\mathcal{R}_{\text{sodh4}}$ in gray.



(b) Game $\texttt{Gagsodh4}^b$, package `Mod-Sodh4` in gray.



(d) Game $\texttt{Gsodh}^b$, package `Mod-Sodh` in gray.

`RdhXtr`

$\mathsf{XTR}_{hs,\ell\in\{0..d\}}(h_1, h_2)$

$n_1, n_2 \leftarrow \mathsf{PrntN}(n)$
$h \leftarrow \mathsf{xtr}\langle hs, h_1, h_2\rangle$
$(k_1, hon_1) \leftarrow \mathsf{GET}_{esalt,\ell}(h_1)$
$h \leftarrow \mathsf{dh}\langle\mathsf{sort}(X, Y)\rangle$
$hon_2 \leftarrow \mathsf{HON}(X) \wedge \mathsf{HON}(Y)$
$k \leftarrow \mathsf{XTR}(h_2, k_1)$
$hon \leftarrow hon_1 \vee hon_2$
$h \leftarrow \mathsf{SET}_{hs,\ell}(h, hon, k)$
return h

(c) Code of `RdhXtr`.

Fig. 36: Reduction $\mathcal{R}_{\text{sodh4}}$ and packages `RdhXtr`, `Mod-Sodh4`, `Mod-Sodh` for the proof of Lemma E.2

Proof of Lemma E.2 Reduction $\mathcal{R}_{\text{sodh4}}$ is defined in Fig. 36a. We need to show that for $b \in \{0, 1\}$

$$\texttt{Gsodh}^b \overset{\text{code}}{\equiv} \mathcal{R}_{\text{sodh4}} \to \texttt{Gagsodh4}^b) \tag{28}$$

and then for all adversaries $\mathcal{A}$,

$$\mathsf{Adv}(\mathcal{A}, \texttt{Gsodh}^b) \overset{(28)}{=} \mathsf{Adv}(\mathcal{A}, \mathcal{R}_{\text{sodh4}} \to \texttt{Gagsodh4}^b) = \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{\text{sodh4}}, \texttt{Gagsodh4}^b)$$

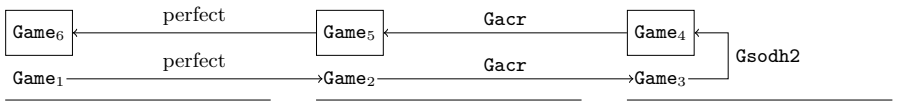
and Lemma E.2 follows. Thus, it suffices to prove Equation 28. In Fig. 36b, we decompose $\texttt{Gagsodh4}^b$ into several packages, and in Fig. 36d, we decompose $\texttt{Gsodh}^b$ into several packages. Observe that the package graphs in Fig. 36b and Fig. 36d are identical except for $\texttt{Mod-Sodh4}^b$ and $\texttt{Mod-Sodh}^b$. Thus, in Fig. 37 and Fig. 38, we provide inlined versions of $\texttt{Mod-Sodh4}^b$ and $\texttt{Mod-Sodh}^b$ and compare their behaviour. See the captions of Fig. 37 and Fig. 38 for the remaining steps to of the proof of Eq. 28.

$\texttt{Mod-Sodh} := \frac{\texttt{DH}}{\texttt{Xtr}} \to \texttt{Nkey} \to \texttt{Log}$	$\texttt{Gagsodh4}^b$
$\mathsf{DHEXP}(X, Y)$	$\mathsf{DHEXP}(X, Y)$
assert $\mathsf{grp}(X) = \mathsf{grp}(Y)$	**assert** $\mathsf{grp}(X) = \mathsf{grp}(Y)$
$h \leftarrow \mathsf{dh}\langle\mathsf{sort}(X, Y)\rangle$	$h \leftarrow \mathsf{dh}\langle\mathsf{sort}(X, Y)\rangle$
$hon_X \leftarrow E[X] \neq \bot$	$hon_X \leftarrow E[X] \neq \bot$
$hon_Y \leftarrow E[Y] \neq \bot$	$hon_Y \leftarrow E[Y] \neq \bot$
assert $hon_X = 1$	**assert** $hon_X = 1$
$x \leftarrow E[X]; k \leftarrow Y^x$	$x \leftarrow E[X]; k \leftarrow Y^x$
$hon \leftarrow hon_X \wedge hon_Y$	$hon \leftarrow hon_X \wedge hon_Y$
assert $\mathsf{name}(h) = dh$	
if $\mathrm{Log}[h] \neq \bot$:	**if** $\mathrm{Log}[h] \neq \bot$:
$(hon, k, h') \leftarrow \mathrm{Log}[h]$	$(hon, k, h') \leftarrow \mathrm{Log}[h]$
return h'	**return** h'
if $(\exists\, h' : \mathrm{Log}_n[h'] = (h', hon', k)$	**if** $(\exists\, h' : \mathrm{Log}_n[h'] = (h', hon', k)$
$\wedge\, \mathsf{level}(h) = r \wedge \mathsf{level}(h^\star) = r')$:	
if $map(r, hon, r', hon' J_n[k])$:	**if** $hon = hon' = 0$:
$\mathrm{Log}_n[h] \leftarrow (h', hon, k)$	$\mathrm{Log}[h] \leftarrow (hon, k, h')$
$\mathrm{J}_n[k] \leftarrow 1$	
$h'' \leftarrow h'$	**return** h'
else	
if $(\exists\, h^\star : \mathrm{Log}_n[h^\star] = (h', hon', k)$	
$\wedge\, \mathsf{level}(h) = r \wedge \mathsf{level}(h^\star) = r')$:	
$P(r, hon, r', hon')$	
$Log_n[h] \leftarrow (h, hon, k)$	$\mathrm{Log}[h] \leftarrow (hon, k, h)$
$h'' \leftarrow h$	
if $h'' \neq h$:	
return h''	
$K_n[h] \leftarrow (k, hon)$	$K[h] \leftarrow k$
return h	**return** h

Fig. 37: $\texttt{Mod-Sodh} \overset{\text{func}}{=} \texttt{Gagsodh4}^b$ for the DHEXP oracle. The observation here is that (a) the Log package has an ∞ map and an Z pattern which simplifies the code and the name of the handle is always correct.

$\texttt{Mod-Sodh} := \frac{\texttt{DH}}{\texttt{Xtr}} \to \texttt{Nkey} \to \texttt{Log}$	$\texttt{Mod-Sodh-Hyb}$	$\texttt{RdhXtr} \to \texttt{Gagsodh4}^b$
$\mathsf{XTR}_{hs,\ell}(h_1, h_2)$	$\mathsf{XTR}_{hs,\ell}(h_1, h_2)$	$\mathsf{XTR}_{hs,\ell}(h_1, h_2)$
$n_1, n_2 \leftarrow \mathsf{PrntN}(hs)$	$n_1, n_2 \leftarrow \mathsf{PrntN}(hs)$	
if $\mathsf{alg}(h_1) \neq \bot \wedge \mathsf{alg}(h_2) \neq \bot$:		
assert $\mathsf{alg}(h_1) = \mathsf{alg}(h_2)$		
$h \leftarrow \mathsf{xtr}\langle n, h_1, h_2\rangle$	$h \leftarrow \mathsf{xtr}\langle n, h_1, h_2\rangle$	$h \leftarrow \mathsf{xtr}\langle hs, h_1, h_2\rangle$
$(k_1, hon_1) \leftarrow \mathsf{GET}_{n_1,\ell}(h_1)$	$(k_1, hon_1) \leftarrow \mathsf{GET}_{n_1,\ell}(h_1)$	$(k_1, hon_1) \leftarrow \mathsf{GET}_{esalt,\ell}(h_1)$
		$\mathsf{dh}\langle\mathsf{sort}(X, Y)\rangle \leftarrow h_2$
		$hon_2 \leftarrow L(X) \neq \bot \wedge L(Y) \neq \bot$
assert $K_{dh}[h_2] \neq \bot$	**assert** $K_{dh}[h_2] \neq \bot$	**assert** $K[h_2] \neq \bot$
$(k^*, hon_2) \leftarrow K_{dh}[h_2]$	$(k^*, hon_2) \leftarrow K_{dh}[h_2]$	**assert** $\mathsf{alg}(k_1) \in \mathcal{H}$
$k_2 \leftarrow \mathsf{tag}_h(k^*)$	$k_2 \leftarrow \mathsf{tag}_h(k^*)$	$\mathsf{dh}\langle\mathsf{sort}(X, Y)\rangle \leftarrow h_2$
$k \leftarrow \mathsf{xtr}(k_1, k_2)$		**if** $\neg b \vee L[X] = \bot$
$hon \leftarrow hon_1 \vee hon_2$		$\vee\, L[Y] = \bot$:
	$k \leftarrow \mathsf{xtr}(k_1, k_2)$	$k^* \leftarrow \mathsf{xtr}\text{-}alg(k_1, K[h_2])$
if $b \wedge hon_2$:	**if** $b \wedge hon_2$:	**else**
		if $S[h_2, k_1] \neq \bot$:
$k^\star \leftarrow\!\!\$\ \{0,1\}^{\mathsf{len}(k)}$	$k^\star \leftarrow\!\!\$\ \{0,1\}^{\mathsf{len}(k)}$	$S[h_2, k_1] \leftarrow\!\!\$\ \{0,1\}^{\mathsf{len}(\mathsf{alg}(k_1)))}$
		$k^* \leftarrow S[h_2, k_1]$
$k \leftarrow \mathsf{tag}_{\mathsf{alg}(k)}(k^\star)$	$k \leftarrow \mathsf{tag}_{\mathsf{alg}(k_1)}(k^\star)$	$k \leftarrow \mathsf{tag}_{\mathsf{alg}(k_1)}(k^*)$
	$hon \leftarrow hon_1 \vee hon_2$	$hon \leftarrow hon_1 \vee hon_2$
$h \leftarrow \mathsf{SET}_{hs,\ell}(h, hon, k)$	$h \leftarrow \mathsf{SET}_{hs,\ell}(h, hon, k)$	$h \leftarrow \mathsf{SET}_{hs,\ell}(h, hon, k)$
return h	**return** h	**return** h

Fig. 38: In Column 1, the inlined `Nkey` package is marked in blue. From Column 1 to 2 the honesty and key calculation (marked in pink) is moved further down. From Column 2 to 3 is slightly more complex. First observe that in Column 3 the honesty of the Diffie-Hellmann shares is recomputed (the first such computation is marked in pink). Secondly the XTR code inlined from `Gagsodh4` (displayed in blue) operates on untagged keys and uses the non-agile xtr function. Algorithm tags are re-added by the reduction code. While the code in the last column utilizes the S table to store randomly sampled values to give consistent oracle answers, the code in column 2 relies in the fact that the SET oracle will discard the value of a second invocation with the same handle h (due to the call to the Q oracle). Finally observe that the **assert** $\mathsf{alg}(k_1) \in \mathcal{H}$ is guaranteed to succeed as the algorithm of the *esalt* was already verified upon storage in the `Key` package.



$\mathtt{Game}_1$	$\mathtt{Game}_2$	$\mathtt{Game}_3$
parameter: $grp, b = 0$ $\boxed{b = 1}$	parameter: $grp, b = 0$ $\boxed{b = 1}$	parameter: $grp, b = 0$ $\boxed{b = 1}$
DHGEN(grp)	DHGEN(grp)	DHGEN(grp)
assert $\mathsf{grp} \in \mathcal{G}$	**assert** $\mathsf{grp} \in \mathcal{G}$	**assert** $\mathsf{grp} \in \mathcal{G}$
$g \leftarrow \mathsf{gen}(grp)$	$g \leftarrow \mathsf{gen}(grp)$	$g \leftarrow \mathsf{gen}(grp)$
$q \leftarrow \mathsf{ord}(g)$	$q \leftarrow \mathsf{ord}(g)$	$q \leftarrow \mathsf{ord}(g)$
$x \leftarrow_\$ Z_q$	$x \leftarrow_\$ Z_q$	$x \leftarrow_\$ Z_q$
$X \leftarrow g^x$	$X \leftarrow g^x$	$X \leftarrow g^x$
$L[X] \leftarrow x$	$L[X] \leftarrow x$	$L[X] \leftarrow x$
return X	**return** X	**return** X
DHEXP(X, Y)	DHEXP(X, Y)	DHEXP(X, Y)
assert $\mathsf{grp}(X) = \mathsf{grp}(Y)$	**assert** $\mathsf{grp}(X) = \mathsf{grp}(Y)$	**assert** $\mathsf{grp}(X) = \mathsf{grp}(Y)$
$h \leftarrow \mathsf{dh}\langle\mathsf{sort}(X, Y)\rangle$	$h \leftarrow \mathsf{dh}\langle\mathsf{sort}(X, Y)\rangle$	$h \leftarrow \mathsf{dh}\langle\mathsf{sort}(X, Y)\rangle$
$hon_X \leftarrow E[X] \neq \bot$	$hon_X \leftarrow E[X] \neq \bot$	$hon_X \leftarrow E[X] \neq \bot$
$hon_Y \leftarrow E[Y] \neq \bot$	$hon_Y \leftarrow E[Y] \neq \bot$	$hon_Y \leftarrow E[Y] \neq \bot$
assert $hon_X = 1$	**assert** $hon_X = 1$	**assert** $hon_X = 1$
$x \leftarrow E[X]$;	$x \leftarrow E[X]$;	$x \leftarrow E[X]$;
$k \leftarrow Y^x$	$k \leftarrow Y^x$	$k \leftarrow \mathsf{xtr}\text{-}alg(0salt, Y^x)$
$hon \leftarrow hon_X \wedge hon_Y$	$hon \leftarrow hon_X \wedge hon_Y$	$hon \leftarrow hon_X \wedge hon_Y$
if $\mathrm{Log}[h] \neq \bot$:	**if** $\mathrm{Log}[h] \neq \bot$:	**if** $\mathrm{Log}[h] \neq \bot$:
$(hon, k, h') \leftarrow \mathrm{Log}[h]$	$(hon, k, h') \leftarrow \mathrm{Log}[h]$	$(hon, k, h') \leftarrow \mathrm{Log}[h]$
return h'	**return** h'	**return** h'
if $(\exists\, h' : \mathrm{Log}[h'] = (h', hon', k)$	**if** $(\exists\, h' : \mathrm{Log}[h'] = (h', hon', k)$	**if** $(\exists\, h' : \mathrm{Log}[h'] = (h', hon', k)$
if $hon = hon^* = 0$:	**if** $hon = hon^* = 0$:	**if** $hon = hon^* = 0$:
$\mathrm{Log}[h] \leftarrow (h', hon, k)$	$\mathrm{Log}[h] \leftarrow (h', hon, k)$	$\mathrm{Log}[h] \leftarrow (h', hon, k)$
return h'	**return** h'	**return** h'
$Log[h] \leftarrow (h, hon, k)$	$Log[h] \leftarrow (h, hon, k)$	$Log[h] \leftarrow (h, hon, k)$
$K[h] \leftarrow k$	$K[h] \leftarrow 1$	$K[h] \leftarrow 1$
return h	**return** h	**return** h
XTR($h, salt$)	XTR($h, salt$)	XTR($h, salt$)
assert $K[h] \neq \bot$	**assert** $K[h] = 1$	**assert** $K[h] = 1$
assert $\mathsf{alg}(salt) \in \mathcal{H}$	**assert** $\mathsf{alg}(salt) \in \mathcal{H}$	**assert** $\mathsf{alg}(salt) \in \mathcal{H}$
$\mathsf{dh}\langle\mathsf{sort}(X, Y)\rangle \leftarrow h$	$\mathsf{dh}\langle\mathsf{sort}(X, Y)\rangle \leftarrow h$	$\mathsf{dh}\langle\mathsf{sort}(X, Y)\rangle \leftarrow h$
if $\neg b \vee L[X] = \bot$	**if** $L[X] = \bot$:	**if** $L[X] = \bot$:
$\vee\, L[Y] = \bot$:	$X, Y \leftarrow \mathsf{swap}(X, Y)$	$X, Y \leftarrow \mathsf{swap}(X, Y)$
return $\mathsf{xtr}\text{-}alg(salt, K[h])$	**if** $L[Y] = \bot \vee \neg b$:	**if** $L[Y] = \bot \vee \neg b$:
if $S[h, salt] \neq \bot$:	**return** $\mathsf{xtr}\text{-}alg(salt, Y^{L[X]})$	**return** $\mathsf{xtr}\text{-}alg(salt, Y^{L[X]})$
$S[h, salt] \leftarrow_\$ \{0,1\}^{\mathsf{len}(\mathsf{alg}(salt))}$	**if** $S[h, salt] \neq \bot$:	**if** $S[h, salt] \neq \bot$:
return $S[h, salt]$	$S[h, salt] \leftarrow_\$ \{0,1\}^{\mathsf{len}(\mathsf{alg}(salt))}$	$S[h, salt] \leftarrow_\$ \{0,1\}^{\mathsf{len}(\mathsf{alg}(salt))}$
	return $S[h, salt]$	**return** $S[h, salt]$

Fig. 39: Game hops relating $\mathtt{Gagsodh4}^b$ and $\mathtt{Gsodh2}^b$

Proof of Lemma E.3 The proof proceeds in 5 game hops, depicted in Fig. 39. Let $\mathcal{A}$ be an adversary. First, observe that $\mathtt{Game}_1 \equiv \mathtt{Gagsodh4}^0$ and $\mathtt{Game}_6 \equiv \mathtt{Gagsodh4}^1$, i.e.,

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gagsodh4}^0, \mathtt{Gagsodh4}^1) = \mathsf{Adv}(\mathcal{A}, \mathtt{Game}_1, \mathtt{Game}_6).$$

From $\mathtt{Game}_1$ to $\mathtt{Game}_2$ (and from $\mathtt{Game}_6$ to $\mathtt{Game}_5$), we replace reading $K[h]$ in XTR with its re-computation. As the same value is computed again, this change is immaterial. Note that the values stored in $K[h]$ are now not used anywhere anymore and we replace them with 1. Thus,

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Game}_1, \mathtt{Game}_6) = \mathsf{Adv}(\mathcal{A}, \mathtt{Game}_2, \mathtt{Game}_5).$$

From $\mathtt{Game}_2$ to $\mathtt{Game}_3$ (and from $\mathtt{Game}_5$ to $\mathtt{Game}_4$), we replace line $k \leftarrow Y^x$ of the DHEXP oracle with $k \leftarrow \mathsf{xtr}\text{-}alg(\mathit{0salt}, Y^x)$. Unless the adversary queries DHEXP on values X, Y and X', Y', such that $Y^x \neq Y'^{x'}$ but $\mathsf{xtr}\text{-}alg(\mathit{0salt}, Y^x) = \mathsf{xtr}\text{-}alg(\mathit{0salt}, Y'^{L[X']})$ this change is perfectly indistinguishable. Formally, this step reduces collision-resistance, i.e., for $z \in \{2, 5\}$, let reduction $\mathcal{R}^{\mathrm{cr},z}_{\mathrm{sodh2}}$ be the reduction which emulates game $\mathtt{Game}_6$ and forwards all xtr calls to its own HASH oracle. We encode both reductions $\mathcal{R}^{\mathrm{cr},z}_{\mathrm{sodh2}}$ for $z \in \{2, 5\}$ into a single reduction $\mathcal{R}^{\mathrm{cr}}_{\mathrm{sodh2}}$ which starts by sampling $z \leftarrow_{\$} \{2, 5\}$ and then runs $\mathcal{R}^{\mathrm{cr},z}_{\mathrm{sodh2}}$. We obtain

$$\begin{aligned} &\mathsf{Adv}(\mathcal{A}, \mathtt{Game}_2, \mathtt{Game}_5) \qquad (29)\\ \leq &\mathsf{Adv}(\mathcal{A}, \mathtt{Game}_3, \mathtt{Game}_4) + 2 \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{alg}_{\text{sodh2-cr}}, \mathtt{Gcr}^{\mathsf{xtr}\text{-}alg,b}) \end{aligned}$$

Inequality 29 holds for all $alg \in \mathcal{H}$. Thus, we can take the minimum over $alg \in \mathcal{H}$ and obtain

$$\begin{aligned} &\mathsf{Adv}(\mathcal{A}, \mathtt{Game}_2, \mathtt{Game}_5)\\ \leq &\mathsf{Adv}(\mathcal{A}, \mathtt{Game}_3, \mathtt{Game}_4) + 2 \cdot \min_{alg \in \mathcal{H}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{alg}_{\text{sodh2-cr}}, \mathtt{Gcr}^{\mathsf{xtr}\text{-}alg,b}) \end{aligned}$$

Finally, we reduce the difference between $\mathtt{Game}_3$ and $\mathtt{Game}_4$ to $\mathtt{Gagsodh2}^b$. I.e., we define reduction $\mathcal{R}_{\mathrm{sodh2}}$ in Fig. 40 such that

$$\mathtt{Game}_3 \overset{\text{func}}{\equiv} \mathtt{Gagsodh2}^0 \text{ and } \mathtt{Game}_4 \overset{\text{func}}{\equiv} \mathtt{Gagsodh2}^1. \qquad (30)$$

and thus,

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Game}_3, \mathtt{Game}_4) \leq \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{\mathrm{sodh2}}, \mathtt{Gagsodh2}^0, \mathtt{Gagsodh2}^1) \qquad (31)$$

Where the equivalence of $\mathcal{R}_{\mathrm{sodh2}}, \mathtt{Gagsodh2}^b$ and $\mathtt{Game}_{3/4}$ follows by simulating the honesty oracle HON in the following way: For each call to DHGEN the reduction forwards the call to $\mathtt{Gagsodh2}^b$ and then marks the returned share as honest in its honesty table H all further honesty checks can then be answered using this table. In addition, the calls to the xtr functions are replaced by calls to the XTR oracle of $\mathtt{Gagsodh2}^b$ and thus no longer need access to the secret exponents. The call to xtr in DHEXP needs to use a tagged *0salt* value for this to work. Observe also that the call to the xtr and equivalently XTR only is executed for dishonest shares whereas the call in the XTR oracle of $\mathtt{Game}_{3/4}$ needs to take care of both the honest and dishonest case.

$\mathtt{Game}_{i\in\{3,4\}}$ $\quad b \in \{0,1\}$

DHGEN()

assert $grp \in \mathcal{G}$
$g \leftarrow \mathsf{gen}(grp)$
$q \leftarrow \mathsf{ord}(g)$
$x \leftarrow\!\!\$\ Z_q$
$X \leftarrow g^x$
$E[X] \leftarrow x$
return X

DHEXP(X, Y)

assert $\mathsf{grp}(X) = \mathsf{grp}(Y)$
$h \leftarrow \mathsf{dh}\langle\mathsf{sort}(X,Y)\rangle$
$hon_X \leftarrow E[X] \neq \bot$
$hon_Y \leftarrow E[Y] \neq \bot$
assert $hon_X = 1$
$x \leftarrow E[X]$;
$k \leftarrow \mathsf{xtr}\text{-}alg(0salt, Y^x)$
$hon \leftarrow hon_X \wedge hon_Y$
if $\mathrm{Log}[h] \neq \bot$:
  $(hon, k, h') \leftarrow \mathrm{Log}[h]$
  return h'
if $(\exists\ h' : \mathrm{Log}_n[h'] = (h', hon', k)$
  if $hon = hon^* = 0$:
    $\mathrm{Log}_n[h] \leftarrow (h', hon, k)$
    return h'
$Log_n[h] \leftarrow (h, hon, k)$
$K_n[h] \leftarrow 1$
return h

XTR($h, salt$)

assert $K[h] = 1$
assert $\mathsf{alg}(salt) \in \mathcal{H}$
$\mathsf{dh}\langle\mathsf{sort}(X,Y)\rangle \leftarrow h$
if $E[X] = \bot$:
  $X, Y \leftarrow \mathsf{swap}(X, Y)$
if $E[Y] = \bot \vee \neg b$:
  return $\mathsf{xtr}\text{-}alg(salt, Y^{L[X]})$
if $S[h, salt] \neq \bot$:
  $S[h, salt] \leftarrow\!\!\$\ \{0,1\}^{\mathsf{len}(\mathsf{alg}(salt))}$
return $S[h, salt]$

HON(X)

return $E[X] \neq \bot$

$\mathcal{R}_{\mathrm{sodh2}}$

DHGEN(grp)

$X \leftarrow$ DHGEN(grp)
$H[X] \leftarrow 1$
return X

DHEXP(X, Y)

assert $\mathsf{grp}(X) = \mathsf{grp}(Y)$
$h \leftarrow \mathsf{dh}\langle\mathsf{sort}(X,Y)\rangle$
$hon_X \leftarrow H[X]$
$hon_Y \leftarrow H[X]$
assert $hon_X = 1$
$k \leftarrow \mathsf{XTR}(X, Y, \mathsf{tag}_{alg}(0salt))$
$hon \leftarrow hon_X \wedge hon_Y$
if $\mathrm{Log}[h] \neq \bot$:
  $(hon, k, h') \leftarrow \mathrm{Log}[h]$
  return h'
if $(\exists\ h' : \mathrm{Log}_n[h'] = (h', hon', k)$
  if $hon = hon^* = 0$:
    $\mathrm{Log}_n[h] \leftarrow (h', hon, k)$
    return h'
$Log_n[h] \leftarrow (h, hon, k)$
$K_n[h] \leftarrow 1$
return h

XTR($h, salt$)

assert $K[h] = 1$
assert $\mathsf{alg}(salt) \in \mathcal{H}$
$\mathsf{dh}\langle\mathsf{sort}(X,Y)\rangle \leftarrow h$
if $H[X] \neq 1$
  $(X, Y) \leftarrow \mathsf{swap}(X, Y)$
$k \leftarrow \mathsf{XTR}(X, Y, salt)$
return k

HON(X)

return $H[X]$

$\mathtt{Gagsodh2}^b$

DHGEN(grp)

assert $grp \in \mathcal{G}$
$g \leftarrow \mathsf{gen}(grp)$
$q \leftarrow \mathsf{ord}(g)$
$x \leftarrow\!\!\$\ Z_q$
$X \leftarrow g^x$
$L[X] \leftarrow x$
return X

XTR($X, Y, salt$)

assert $L[X] \neq \bot$
  $\wedge\ \mathsf{grp}(X) = \mathsf{grp}(Y)$
  $\wedge\ \mathsf{alg}(salt) \in \mathcal{H}$
$alg \leftarrow \mathsf{alg}(salt)$
if $b \wedge L[Y] \neq \bot$:
  $h \leftarrow \mathsf{dh}\langle\mathsf{sort}(X,Y)\rangle$
  if $S[h, salt] = \bot$:
    $S[h, salt] \leftarrow\!\!\$\ \{0,1\}^{\mathsf{len}(alg)}$
  return $S[h, salt]$
return $\mathsf{xtr}\text{-}alg(salt, Y^{L[X]})$

Fig. 40: $\mathcal{R}_{\mathrm{sodh2}} \rightarrow \mathtt{Gagsodh2}^0 \overset{\mathrm{code}}{=} \mathtt{Game}_3$ (left), $\mathcal{R}_{\mathrm{sodh2}} \rightarrow \mathtt{Gagsodh2}^1 \overset{\mathrm{code}}{=} \mathtt{Game}_4$ (middle) and $\mathtt{Gsodh2}^b$ (right).

Proof of Lemma E.4 Let $<$ be an arbitrary order on $\mathcal{G}$ with grp_{fst} being the smallest group in $\mathcal{G}$ with respect to the order $<$, let grp_{lst} being the largest group in $\mathcal{G}$ with respect to the order $<$ and for $grp < grp_{\text{lst}}$, let $grp+1$ be the direct successor of grp according to the order $<$. The proof proceeds via a hybrid argument over $grp \in \mathcal{G}$ in the order $<$. For reduction $\mathcal{R}_{grp}$ (Fig. 41), we have the following equalities:

$$\mathcal{R}_{grp_{\text{fst}}} \to \texttt{Gsodh2}^{grp_{\text{fst}},0} \overset{\text{code}}{\equiv} \texttt{Gagsodh2}^{0} \tag{32}$$

$$\mathcal{R}_{grp_{\text{lst}}} \to \texttt{Gsodh2}^{grp_{\text{lst}},1} \overset{\text{code}}{\equiv} \texttt{Gagsodh2}^{1} \tag{33}$$

$$\mathcal{R}_{grp} \to \texttt{Gsodh2}^{grp,1} \overset{\text{code}}{\equiv} \mathcal{R}_{grp+1} \to \texttt{Gsodh2}^{grp+1,0} \tag{34}$$

where $grp < grp_{\text{lst}}$. Given Equations (32)-(34), we can then use a telescopic sum and the triangle inequality, as is standard for for hybrid arguments.

$$\begin{aligned}
&\mathsf{Adv}(\mathcal{A}, \to \texttt{Gagsodh2}^{b}) \\
\overset{(32)+(33)}{=}\ &\big|\Pr\big[1 \leftarrow \mathcal{A} \to \mathcal{R}_{grp_{\text{fst}}} \to \texttt{Gsodh2}^{grp_{\text{fst}},0}\big] \\
&\quad - \Pr\big[1 \leftarrow \mathcal{A} \to \mathcal{R}_{grp_{\text{lst}}} \to \texttt{Gsodh2}^{grp_{\text{lst}},1}\big]\big| \\
\overset{(34)}{=}\ &\Bigg|\sum_{grp \in \mathcal{G}} \Pr\big[1 \leftarrow \mathcal{A} \to \mathcal{R}_{grp} \to \texttt{Gsodh2}^{grp,0}\big] \\
&\quad - \Pr\big[1 \leftarrow \mathcal{A} \to \mathcal{R}_{grp} \to \texttt{Gsodh2}^{grp,1}\big]\Bigg| \\
\leq\ &\sum_{grp \in \mathcal{G}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{grp}, \texttt{Gsodh2}^{grp,b})
\end{aligned}$$

This concludes the proof of Lemma E.4.

$\underline{\mathcal{R}_{grp}}$

$\underline{\mathsf{DHGEN}(grp^*)}$

if $grp = grp^*$:
  return $\mathsf{DHGEN}()$
assert $grp \in \mathcal{G}$
$g \leftarrow \mathsf{gen}(grp^*)$
$q \leftarrow \mathsf{ord}(g)$
$x \leftarrow_{\$} Z_q$
$X \leftarrow g^x$
$L[X] \leftarrow x$
return X

$\underline{\mathsf{XTR}(X, Y, salt)}$

$grp^* \leftarrow \mathsf{grp}(X)$
if $grp^* = grp$
  return $\mathsf{XTR}(X, Y, salt)$
assert $L[X] \neq \bot$
  $\wedge\ \mathsf{grp}(X) = \mathsf{grp}(Y)$
  $\wedge\ \mathsf{alg}(salt) \in \mathcal{H}$
$alg \leftarrow \mathsf{alg}(salt)$
if $grp^* > grp \wedge L[Y] \neq \bot$:
  $h \leftarrow \mathsf{dh}\langle\mathsf{sort}(X, Y)\rangle$
  if $S[h, salt] = \bot$:
    $S[h, salt] \leftarrow_{\$} \{0,1\}^{\mathsf{len}(alg)}$
  return $S[h, salt]$
return $\mathsf{xtr}\text{-}alg(salt, Y^{L[X]})$

Fig. 41: Oracles of reduction $\mathcal{R}_{grp}$.

E.2 Generic Games

In this subsection, we introduce a parametrized, generic pseudorandomness game $\texttt{G-M-Pr-io}^{in,\mathsf{f},b,coll_{\text{in}},coll_{\text{Out}}}_{Out}$ (described in Figure 42a and Figure 42c) for an (agile) function f, input name in, output names Out, idealization bit b, input collision resistance pattern $coll_{\text{in}}$, and output collision resistance pattern $coll_{\text{Out}}$. These patterns are used in the input $\texttt{Key}^{1,coll_{\text{in}}}_{in}$ package and output $\texttt{Key}^{b,coll_{\text{Out}}}_{Out}$ package of $\texttt{G-M-Pr-io}^{in,\mathsf{f},b,coll_{\text{in}},coll_{\text{Out}}}_{Out}$. We show in a companion paper [29] that the generic pseudorandomness game reduces to the monolithic non-agile pseudorandomness game $\texttt{Gpr}^{\mathsf{f}\text{-}alg,b}$ introduced in Section 3 and re-state this result here in Lemma E.5. We parametrize the game

$\texttt{G-M-Pr-io}^{in,\mathsf{f},b,coll_{\mathrm{in}},coll_{\mathrm{Out}}}_{Out}$ by an injective handle constructor $\mathsf{inj}\langle\cdot\rangle$ and a function $\mathsf{OutIndex} : dom\,\mathsf{inj}\langle\cdot\rangle \to Out$ which maps a handle h constructed using $\mathsf{inj}\langle\cdot\rangle$ to $(\mathsf{name}(h), \mathsf{level}(h))$.

Lemma E.5 (Monolithic to Composite). *Let $\mathcal{H}$ be a set of algorithms, where $\forall alg, alg' \in \mathcal{H}.\ alg \neq alg' \implies \forall x. |alg(x)| \neq |alg'(x)|$. Let the $\mathcal{H}$-agile function f and the non-agile function f-alg be as described in Fig. 42b. Let in be an index, and let Out be a set of indices. Further let $coll_{in}, coll_{Out} \in \{Z\infty, D, D1, R, F\}$. Then for all adversaries $\mathcal{A}$ against $\texttt{G-M-Pr-io}^{\mathsf{f},b}$, we have*

$$\mathsf{Adv}(\mathcal{A},\texttt{G-M-Pr-io}^{in,\mathsf{f},b,coll_{in},coll_{Out}}_{Out}) \leq$$
$$\sum_{alg\in\mathcal{H}} n_{alg} \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\mathsf{f}\text{-}alg,coll_{in},coll_{Out}}_{\texttt{G-M-Pr-io}}, \texttt{Gpr}^{\mathsf{f}\text{-}alg,b}) + (c+1) \cdot \frac{n^2_{alg}}{2^{\mathsf{len}(alg)}}, \quad (35)$$

where $\mathcal{R}^{\mathsf{f}\text{-}alg,coll_{in},coll_{Out}}_{\texttt{G-M-Pr-io}} := \mathcal{R}^{coll_{Out}}_{\mathtt{o},Out} \to \mathcal{R}^{coll_{in}}_{\{F,R\}} \to \mathcal{R}^{coll_{in}\setminus\{F,R\}}_{\mathtt{i}} \to \mathcal{R}_{\mathtt{alg},alg} \to \mathcal{R}_{\mathtt{M}} \to \mathcal{R}_{\mathtt{Gpr}}$ is defined in [29], $n_{alg,hon=1}$ is the number of $\mathsf{SET}_{in}(h,1,*)$ queries which the adversary makes for handles h with $\mathsf{alg}(h) = alg$, $n_{alg,hon=0}$ is the number of $\mathsf{SET}_{in}(h,0,*)$ queries which the adversary makes for handles h with $\mathsf{alg}(h) = alg$, $n_{alg} = \max\{n_{alg,hon=1}, n_{alg,hon=0}\}$ and c is a small constant which depends on the min-entropy of the distribution $f(k, U_{\mathsf{len}(alg)})$, where k is a fixed key and $U_{2^{\mathsf{len}(alg)}}$ denotes the uniform distribution of strings of length $\mathsf{len}(alg)$. The pattern of reduction $\mathcal{R}^{coll_{in}}_{\{F,R\}}$ depends on $coll_{in}$. If $coll_{in} \notin \{R, F\}$, $\mathcal{R}^{coll_{in}}_{\{F,R\}}$ and the final term can be omitted.

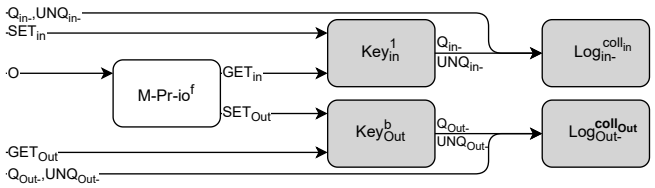
See Fig. 8 for the oracle descriptions of $\texttt{Gpr}^{\mathsf{f}\text{-}alg,b}$ and Fig. 42a and Fig. 42c for package composition and oracle descriptions of $\texttt{G-M-Pr-io}^{\mathsf{f},b}$.

Lemma E.5 takes care of multi-to-single instance reductions and agile-to-non-agile reductions. Note that we treated SODH separately, since (a) the base assumption remains agile (since the same Diffie-Hellman secrets are processed by different algorithms) and (b) the $\texttt{G-M-Pr-io}^{in,\mathsf{f},b,coll_{\mathrm{in}},coll_{\mathrm{Out}}}_{Out}$ game does not account for exponentiation/DH operations.

In Appendix A.1, we reduced agile to non-agile collision-resistance for algorithms with disjoint output domains. It remains to provide reductions for pseudorandomness games and pre-image resistance games to $\texttt{G-M-Pr-io}^{in,\mathsf{f},b,coll_{\mathrm{in}},coll_{\mathrm{Out}}}_{Out}$, which we carry out in Appendix E.3 and Appendix E.4, respectively.

E.3 Pseudorandomness Assumptions

In this appendix, we prove Lemma E.6 (Appendix B), i.e., we upper bound the advantage of an adversary $\mathcal{A}$ playing against the pseudorandomness games $\texttt{Gxtr1}_{es,\ell}$, $\texttt{Gxtr2}_{hs,\ell}$, $\texttt{Gxtr3}_{as,\ell}$, $\texttt{Gxpd}_{n,\ell}$ by the sum of advantages of an adversary $\mathcal{A} \to \mathcal{R}$ against the pseudorandomness game $\texttt{Gpr}^{\mathsf{f}\text{-}alg,b}$ introduced in Section 3 for a suitable reduction $\mathcal{R}$ and summing over the algorithms alg. Recall that the proof of this relation will rely on the generic pseudorandomness game $\texttt{G-M-Pr-io}^{in,\mathsf{f},b,coll_{\mathrm{in}},coll_{\mathrm{Out}}}_{Out}$ which we introduced in Appendix E.2 and



(a) Composed $\texttt{G-M-Pr-io}^{in,\mathsf{f},b,coll_{\mathrm{in}},coll_{\mathrm{Out}}}_{Out}$ Game.

$\mathsf{f}(k, args)$

$alg \leftarrow \mathsf{alg}(k)$
$k \leftarrow \mathsf{untag}(k)$
$k' \leftarrow \mathsf{f}\text{-}alg(k, args)$
return $\mathsf{tag}_{alg}(k')$

(b) Agile function f.

$\underline{\texttt{M-Pr-io}^{\mathsf{f}}}$

Parameters

in	input index
Out	output indices
inj	handle constructor
$\mathsf{OutIndex}$	index function

$\mathsf{O}(h, args)$

$(k, hon) \leftarrow \mathsf{GET}_{in}(h)$
$k_{out} \leftarrow \mathsf{f}(k, args)$
$h_{out} \leftarrow \mathsf{inj}\langle h, args\rangle$
$i \leftarrow \mathsf{OutIndex}(h_{out})$
assert $i \in Out$
return $\mathsf{SET}_i(h_{out}, hon, k_{out})$

(c) Oracles of $\texttt{M-Pr-io}^{\mathsf{f}}$. For Key and Log, see Figure 22.

Fig. 42

Lemma E.5, which relates $\texttt{G-M-Pr-io}^{in,\mathsf{f},b,coll_{\mathrm{in}},coll_{\mathrm{Out}}}_{Out}$ to $\texttt{Gpr}^{\mathsf{f}\text{-}alg,b}$. We now restate Lemma E.6.

Lemma E.6 (Advantages). *Let $n \in XPR$ and let $(n_1, \_) = \mathsf{PrntN}(n)$. Let $\mathcal{A}$ be an adversary that generates at most $s_{n,\ell,alg}$ honest keys for algorithm alg with name n_1 at level ℓ via $\mathsf{SET}_{n_1,\ell}(*, 1, *)$ queries. The* xpd *pseudorandomness advantage is bounded by*

$$\mathsf{Adv}(\mathcal{A}, \texttt{Gxpd}^0_{n,\ell}, \texttt{Gxpd}^1_{n,\ell}) \leq \sum_{alg \in \mathcal{H}} s_{n,\ell,alg} \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{alg}_{n,\ell}, \texttt{Gpr}^{\mathsf{xpd}\text{-}alg,b}) \tag{6}$$

$$\mathsf{Adv}(\mathcal{A}, \texttt{Gxpd}^0_{psk,\ell}, \texttt{Gxpd}^1_{psk,\ell}) \leq \sum_{alg \in \mathcal{H}} s_{psk,\ell,alg} \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{alg}_{psk,\ell}, \texttt{Gpr}^{\mathsf{xpd}\text{-}alg,b}) \tag{7}$$

$$\mathsf{Adv}(\mathcal{A}, \texttt{Gxpd}^0_{esalt,\ell}, \texttt{Gxpd}^1_{esalt,\ell}) \leq \sum_{alg \in \mathcal{H}} s_{esalt,\ell,alg} \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{alg}_{esalt,\ell}, \texttt{Gpr}^{\mathsf{xpd}\text{-}alg,b}) \tag{8}$$

Let $\mathcal{A}$ be an adversary that generates at most $s_{es,\ell,alg}$ honest keys for algorithm alg with name psk at level ℓ via $\mathsf{SET}_{psk,\ell}(*, 1, *)$ queries. The xtr *pseudorandomness advantage is bounded by*

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gxtr1}^0_{es,\ell}, \mathtt{Gxtr1}^1_{es,\ell})$$
$$\leq \sum_{alg \in \mathcal{H}} s_{es,\ell,alg} \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{alg}_{es,\ell}, \mathtt{Gpr}^{\mathsf{xtr}^\dagger\text{-}alg,b}) \tag{9}$$

Let $\mathcal{A}$ be an adversary that generates at most $s_{es,\ell,alg}$ honest keys for algorithm alg with name esalt at level ℓ via $\mathsf{SET}_{esalt,\ell}(*,1,*)$ queries. Let n_{alg} is an upper bound on the sum of UNQ_{esalt} and $\mathsf{SET}_{esalt,\ell}$ queries made by $\mathcal{A}$, and let c be a small constant which depends on the min-entropy of the distribution $f(k, U_{\mathsf{len}(alg)})$, where k is a fixed key and $U_{2^{\mathsf{len}(alg)}}$ denotes the uniform distribution of strings of length $\mathsf{len}(alg)$. The xtr *pseudorandomness advantage is bounded by*

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gxtr2}^0_{hs,\ell}, \mathtt{Gxtr2}^1_{hs,\ell})$$
$$\leq \sum_{alg \in \mathcal{H}} s_{hs,\ell,alg} \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{alg}_{hs,\ell}, \mathtt{Gpr}^{\mathsf{xtr}\text{-}alg,b}) + (c+1) \cdot \frac{n^2_{alg}}{2^{\mathsf{len}(alg)}} \tag{10}$$

Let $\mathcal{A}$ be an adversary that generates at most $s_{as,\ell,alg}$ honest keys for algorithm alg with name hsalt at level ℓ via $\mathsf{SET}_{hsalt,\ell}(*,1,*)$ queries. The xtr *pseudorandomness advantage is bounded by*

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gxtr3}^0_{as,\ell}, \mathtt{Gxtr3}^1_{as,\ell})$$
$$\leq \sum_{alg \in \mathcal{H}} s_{as,\ell,alg} \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{alg}_{as,\ell}, \mathtt{Gpr}^{\mathsf{xtr}\text{-}alg,b}) \tag{11}$$

The reductions are defined as follows:

$$\begin{aligned}
\mathcal{R}^{alg}_{n,\ell} &:= \mathcal{R}^{rmprio,\mathsf{xpd}}_{n,\ell} \to \mathcal{R}^{\mathsf{f}\text{-}alg}_{\mathtt{G\text{-}M\text{-}Pr\text{-}io}}\ ,\ \textit{where}\ f = \mathsf{xpd} \\
\mathcal{R}^{alg}_{psk,\ell} &:= \mathcal{R}^{rmprio,\mathsf{xpd}}_{psk,\ell} \to \mathcal{R}^{\mathsf{f}\text{-}alg}_{\mathtt{G\text{-}M\text{-}Pr\text{-}io}}\ ,\ \textit{where}\ f = \mathsf{xpd} \\
\mathcal{R}^{alg}_{esalt,\ell} &:= \mathcal{R}^{rmprio,\mathsf{xpd}}_{esalt,\ell} \to \mathcal{R}^{\mathsf{f}\text{-}alg}_{\mathtt{G\text{-}M\text{-}Pr\text{-}io}}\ ,\ \textit{where}\ f = \mathsf{xpd} \\
\mathcal{R}^{alg}_{es,\ell} &:= \mathcal{R}^{rmprio,\mathsf{xtr}}_{es,\ell} \to \mathcal{R}^{\mathsf{f}\text{-}alg}_{\mathtt{G\text{-}M\text{-}Pr\text{-}io}}\ ,\ \textit{where}\ f = \mathsf{xtr}^\dagger \\
\mathcal{R}^{alg}_{hs,\ell} &:= \mathcal{R}^{rmprio,\mathsf{xtr}}_{hs,\ell} \to \mathcal{R}^{\mathsf{f}\text{-}alg}_{\mathtt{G\text{-}M\text{-}Pr\text{-}io}}\ ,\ \textit{where}\ f = \mathsf{xtr} \\
\mathcal{R}^{alg}_{as,\ell} &:= \mathcal{R}^{rmprio,\mathsf{xtr}}_{as,\ell} \to \mathcal{R}^{\mathsf{f}\text{-}alg}_{\mathtt{G\text{-}M\text{-}Pr\text{-}io}}\ ,\ \textit{where}\ f = \mathsf{xtr}
\end{aligned}$$

For $\mathcal{R}^{\mathsf{f}\text{-}alg}_{\mathtt{G\text{-}M\text{-}Pr\text{-}io}}$, see Lemma E.5. Reduction $\mathcal{R}^{rmprio,\mathsf{xpd}}_{n,\ell}$ is defined in Fig. 54 (code) and Fig. 50b (graph), $\mathcal{R}^{rmprio,\mathsf{xpd}}_{psk,\ell}$ is defined in Fig. 55 (code) and Fig. 50f (graph), $\mathcal{R}^{rmprio,\mathsf{xpd}}_{esalt,\ell}$ is defined in Fig. 54 (code) and Fig. 50d (graph), $\mathcal{R}^{rmprio,\mathsf{xtr}}_{es,\ell}$ is defined in Fig. 51 (code) and Fig. 49b (graph), $\mathcal{R}^{rmfprio,\mathsf{xtr}}_{hs,\ell}$ is defined in Fig. 52 (code) and Fig. 49d (graph), and $\mathcal{R}^{rmprio,\mathsf{xtr}}_{as,\ell}$ is defined in Fig. 53 (code) and Fig. 49f (graph).

Proof. We now prove Inequalities 6-11, i.e., we show that the indistinguishability games for xtr and xpd can be reduced to the G-M-Pr-io game, defined in Fig. 42a. Roughly, all reductions follow by emulating additional Key packages (for the case of xtr) and the $\mathtt{Hash}^1$ package (for the case of xpd). The conceptually most interesting aspect is the translation between handles. Oracle $\mathsf{O}(h, \mathit{args})$ of $\mathtt{G\text{-}M\text{-}Pr\text{-}io}^{in,\mathrm{f},b,coll_{\mathrm{in}},coll_{\mathrm{Out}}}_{Out}$ constructs a handle as some injective function of h and *args*, and this injective functions will be $\langle n, h, \mathit{args}\rangle$ or $\langle n, \mathit{args}, h\rangle$ in all of the theorems. However, this is not what the Xtr and Xpd packages use. Xtr uses the pair of handles $\langle n, h_1, h_2\rangle$ while $\langle n, h_1, \mathit{args}\rangle$ in G-M-Pr-io corresponds to using the *key* k_2 as *args* in the handle $\langle n, h_1, k_2\rangle$ rather than the *handle* h_2. Due to patterns on the input key packages, however, there is an injective mapping from keys to handles and thus, handles which use keys and handles which use handles bijectively map to one another. A similar argument applies to Xpd which uses as *args* for the handle construction the transcript, while G-M-Pr-io uses the *hashed* transcript digest $d = \mathsf{hash}(t)$. Again, due to the idealization bit of the hash package $\mathtt{Hash}^1$ being 1, the mapping from digests to transcripts is injective (in the system) and handles can use both interchangeably. We now proceed to prove the Inequalities 6-11 individually.

Inequality 6 We define reduction $\mathcal{R}^{\mathrm{rmprio},\mathsf{xpd}}_{\mathrm{n},\ell}$ with the property that for $b \in \{0,1\}$

$$\mathtt{Gxpd}^b_{n,\ell}, \stackrel{\mathrm{code}}{\equiv} \mathcal{R}^{\mathrm{rmprio},\mathsf{xpd}}_{\mathrm{n},\ell} \to \mathtt{G\text{-}M\text{-}Pr\text{-}io}^{(n_I,\ell),\mathsf{xpd},b,D,D_{CN}}_{CN} \tag{36}$$

where we re-state $\mathtt{Gxpd}^b_{n,\ell}$ in Fig 50a for convenience. From there, it follows directly that for all adversaries $\mathcal{A}$,

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gxpd}^b_{n,\ell}) \leq \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\mathrm{rmprio},\mathsf{xpd}}_{\mathrm{n},\ell}, \mathtt{G\text{-}M\text{-}Pr\text{-}io}^{(n_I,\ell),\mathsf{xpd},b,D,D_{CN}}_{CN})$$

and with Inequality 35, Inequality 6 directly follows.

It remains to provide the code equivalence proof of Equation 36. The game $\mathcal{R}^{\mathrm{rmprio},\mathsf{xpd}}_{\mathrm{n},\ell} \to \mathtt{G\text{-}M\text{-}Pr\text{-}io}^b$ is provided in Figure 49f with the code of reduction $\mathcal{R}^{\mathrm{rmprio},\mathsf{xpd}}_{\mathrm{n},\ell}$ provided in Figure 54. As discussed in the high-level overview of the proof, reduction $\mathcal{R}^{\mathrm{rmprio},\mathsf{xpd}}_{\mathrm{n},\ell}$ exports to the adversary handles which rely on transcripts and internally uses handles which rely on digests (hashed transcript). We prove Equation 36 by inlining the O oracle of $\mathtt{M\text{-}Pr\text{-}io}^{\mathsf{xpd}}$ into $\mathcal{R}^{\mathrm{rmprio},\mathsf{xpd}}_{\mathrm{n},\ell}$ from the 1st to the 2nd column of Figure 54, highlighted in pink. $\mathcal{R}^{\mathrm{rmprio},\mathsf{xpd}}_{\mathrm{n},\ell}$ exposes an $\mathsf{XPD}_{\mathrm{n'},\ell}$ oracle for each n' which has the same parents as n. Since n' is prepended in the construction of the handle $\langle n', h_1, \mathit{args}\rangle$, the handle can either be ill-formed or have the name n'. It thus suffices if **assert** $i \in \{(n', \ell)\}$ checks for the pair (n', ℓ). From the middle to the right column, this assert is then omitted entirely, since it subsumed by the first lines of the $\mathsf{SET}_{n',\ell}(h, \mathit{hon}, k)$ which cover **assert** $\mathsf{name}(h) = n'$ and **assert** $\mathsf{level}(h) = \ell$.

Inequality 7 The proof is similar to the proof of Inequality 6, but we include it for completeness. We define reduction $\mathcal{R}^{\mathrm{rmprio},\mathsf{xpd}}_{\mathrm{psk},\ell}$ with the property that for

$b \in \{0,1\}$

$$\mathtt{Gxpd}^b_{psk,\ell}, \overset{\text{code}}{\equiv} \mathcal{R}^{\text{rmprio},\mathsf{xpd}}_{\text{psk},\ell} \rightarrow \mathtt{G\text{-}M\text{-}Pr\text{-}io}^{(rm,\ell),\mathsf{xpd},b,D,D1}_{\{(psk,\ell+1)\}} \tag{37}$$

where we re-state $\mathtt{Gxpd}^b_{psk,\ell}$ in Fig 50e for convenience. From there, it follows directly that for all adversaries $\mathcal{A}$,

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gxpd}^b_{psk,\ell}) \leq \mathsf{Adv}(\mathcal{A} \rightarrow \mathcal{R}^{\text{rmprio},\mathsf{xpd}}_{\text{psk},\ell}, \mathtt{G\text{-}M\text{-}Pr\text{-}io}^{(rm,\ell),\mathsf{xpd},b,D,D1}_{\{(psk,\ell+1)\}})$$

and with Inequality 35, Inequality 7 directly follows.

It remains to provide the code equivalence proof of Equation 37. The game $\mathcal{R}^{\text{rmprio},\mathsf{xpd}}_{\text{psk},\ell} \rightarrow \mathtt{G\text{-}M\text{-}Pr\text{-}io}^{(rm,\ell),\mathsf{xpd},b,D,D1}_{\{(psk,\ell+1)\}}$ is provided in Figure 49f and see Figure 55 for the code of reduction $\mathcal{R}^{\text{rmprio},\mathsf{xpd}}_{\text{psk},\ell}$. In contrast to $\mathcal{R}^{\text{rmprio},\mathsf{xpd}}_{\text{n},\ell}$, the *psk* is a special case as its `xpd` arguments are used unmodified (and in particular no digest is computed). Consequently no handle-mapping is needed. We prove Equation 37 by inlining the O oracle of $\mathtt{M\text{-}Pr\text{-}io}^{\mathsf{xpd}}$ into $\mathcal{R}^{\text{rmprio},\mathsf{xpd}}_{\text{psk},\ell}$ from the 1st to the 2nd column of Figure 55, highlighted in pink. $\mathcal{R}^{\text{rmprio},\mathsf{xpd}}_{\text{psk},\ell}$ exposes an $\mathsf{XPD}_{\text{psk},\ell}$ oracle. Since *psk* is prepended in the construction of the handle $\langle psk, h_1, args\rangle$, the handle can either be ill-formed or have the name *psk*. It thus suffices if **assert** $i \in \{(psk, \ell+1)\}$ checks for the pair $(psk, \ell+1)$. Also note the here, there is a level transition due to the *psk*. From the 2nd to the 3rd column, this assert is then omitted entirely, since it subsumed by the first lines of the $\mathsf{SET}_{psk,\ell+1}(h, hon, k)$ which cover **assert** $\mathsf{name}(h) = psk$ and **assert** $\mathsf{level}(h) = \ell + 1$. From the 3rd to the last column the calculation of the level and the key are swapped, highlighted in pink again which yields the *psk* case of the XPD code as required.

Inequality 8 The proof is analogous to the proof of Inequality 8, we discuss it briefly here. The reduction $\mathcal{R}^{\text{rmprio},\mathsf{xpd}}_{\text{esalt},\ell}$ is defined as $\mathcal{R}^{\text{rmprio},\mathsf{xpd}}_{\text{n},\ell}$ (Figure 54) with $n = esalt$ and satisfies the analogous Equality to Equality 36, i.e., for $b \in \{0,1\}$

$$\mathtt{Gxpd}^b_{esalt,\ell}, \overset{\text{code}}{\equiv} \mathcal{R}^{\text{rmprio},\mathsf{xpd}}_{esalt,\ell} \rightarrow \mathtt{G\text{-}M\text{-}Pr\text{-}io}^{(n_1,\ell),\mathsf{xpd},b,D,(D_{CN\setminus\{esalt\}}||R_{esalt})}_{CN}. \tag{38}$$

We re-state $\mathtt{Gxpd}^b_{esalt,\ell}$ in Fig 50c for convenience. As previously, from Equation 38, it follows directly that for all adversaries $\mathcal{A}$,

$$\begin{aligned}&\mathsf{Adv}(\mathcal{A}, \mathtt{Gxpd}^b_{esalt,\ell})\\ \leq&\mathsf{Adv}(\mathcal{A} \rightarrow \mathcal{R}^{\text{rmprio},\mathsf{xpd}}_{\text{esalt},\ell}, \mathtt{G\text{-}M\text{-}Pr\text{-}io}^{(n_1,\ell),\mathsf{xpd},b,D,(D_{CN\setminus\{esalt\}}||R_{esalt})}_{CN})\end{aligned}$$

and with Inequality 35, Inequality 8 directly follows. The code equivalence proof of Equation 38 is identical to the code equivalence proof of Equation 38 since the difference between the game $\mathtt{G\text{-}M\text{-}Pr\text{-}io}^{(n_1,\ell),\mathsf{xpd},b,D,(D_{CN\setminus\{esalt\}}||R_{esalt})}_{CN}$ and the game $\mathtt{G\text{-}M\text{-}Pr\text{-}io}^{(n_1,\ell),\mathsf{xpd},b,D,D_{CN}}_{CN}$ is merely the R pattern for *esalt* in $\mathtt{Log}_{esalt}$, but since we do not inline the $\mathtt{Log}_{esalt}$ package in the inlining proof, the two inlining proofs are identical. See Figure 54.

Inequality 9 We define reduction $\mathcal{R}^{\text{rmprio,xtr}}_{\text{es},\ell}$ with the property that for $b \in \{0,1\}$

$$\texttt{Gxtr1}^b_{es,\ell}, \overset{\text{code}}{\equiv} \mathcal{R}^{\text{rmprio,xtr}}_{\text{es},\ell} \rightarrow \texttt{G-M-Pr-io}^{(psk,\ell),\mathsf{xtr}^\dagger,b,D1,D}_{\{es\}} \tag{39}$$

where we re-state $\texttt{Gxtr1}^b_{es,\ell}$ in Fig 49a for convenience. From there, it follows directly that for all adversaries $\mathcal{A}$,

$$\mathsf{Adv}(\mathcal{A}, \texttt{Gxtr1}^b_{es,\ell}) \leq \mathsf{Adv}(\mathcal{A} \rightarrow \mathcal{R}^{\text{rmprio,xtr}}_{\text{es},\ell}, \texttt{G-M-Pr-io}^{(psk,\ell),\mathsf{xtr}^\dagger,b,D1,D}_{\{es\}})$$

and with Inequality 35, Inequality 9 directly follows.

It remains to provide the code equivalence proof of Equation 39. The game $\mathcal{R}^{\text{rmprio,xtr}}_{\text{es},\ell} \rightarrow \texttt{G-M-Pr-io}^b$ is provided in Figure 49b with the code of reduction $\mathcal{R}^{\text{rmprio,xtr}}_{\text{es},\ell}$ provided in Figure 51. As discussed in the high-level overview of the proof, reduction $\mathcal{R}^{\text{rmprio,xtr}}_{\text{es},\ell}$ exports to the adversary handles which rely on handles and internally relies on handles which rely on concrete key values. We prove Equation 39 by inlining $\texttt{M-Pr-io}^{\mathsf{xtr}^\dagger}$ into $\mathcal{R}^{\text{rmprio,xtr}}_{\text{es},\ell}$ from the 1st to the 2nd column of Figure 51, highlighted in pink. From the 2nd to third column, we then replace the function $\mathsf{xtr}^\dagger$ by swapping its inputs and applying its dual xtr. Moreover, the composed package now uses $\mathsf{xtr}\langle es, h_1, h_2\rangle$ as handle for the $\mathsf{SET}_{es,\ell}(h, hon, k)$ query instead of $\mathsf{xtr}\langle es, k_1, h_2\rangle$ which is a bijective mapping since $\texttt{Nkey}_{0salt}$ only contains a single handle (see Figure 23b). Additionally, we also remove the $\mathsf{OutIndex}$ computation and the assert that its result be in $\{(es,\ell)\}$. This justified, since the first line of $\mathsf{SET}_{es,\ell}(h, hon, k)$ asserts anyway that $\mathsf{name}(h) = es$ and $\mathsf{level}(h) = \ell$. Finally, since internally and externally, the same handles are used now, the queries $\mathsf{GET}_{es,\ell}$, Q_{es} and UNQ_{es} are directly forwarded. This concludes the proof of Equation 39.

Inequality 10 We define reduction $\mathcal{R}^{\text{rmprio,xtr}}_{\text{hs},\ell}$ with the property that for $b \in \{0,1\}$

$$\texttt{Gxtr2}^b_{hs,\ell}, \overset{\text{code}}{\equiv} \mathcal{R}^{\text{rmprio,xtr}}_{\text{hs},\ell} \rightarrow \texttt{G-M-Pr-io}^{(esalt,\ell),\mathsf{xtr},b,R,D}_{\{hs\}} \tag{40}$$

where we re-state $\texttt{Gxtr2}^b_{es,\ell}$ in Fig 49c for convenience. From there, it follows directly that for all adversaries $\mathcal{A}$,

$$\mathsf{Adv}(\mathcal{A}, \texttt{Gxtr1}^b_{hs,\ell}) \leq \mathsf{Adv}(\mathcal{A} \rightarrow \mathcal{R}^{\text{rmprio,xtr}}_{\text{hs},\ell}, \texttt{G-M-Pr-io}^{(esalt,\ell),\mathsf{xtr},b,R,D}_{\{hs\}})$$

It remains to provide the code equivalence proof of Equation 40. The game $\mathcal{R}^{\text{rmprio,xtr}}_{\text{hs},\ell} \rightarrow \texttt{G-M-Pr-io}^b$ is provided in Figure 49f with the code of reduction $\mathcal{R}^{\text{rmprio,xtr}}_{\text{hs},\ell}$ provided in column 1 of Figure 51. The Q_{hs} and UNQ_{hs} query are defined at the very bottom of the 1st column, highlighting text which is only part of UNQ_{hs} but not of Q_{hs} in blue as in UNQ. As in the proof of Equation 39, the reduction $\mathcal{R}^{\text{rmprio,xtr}}_{\text{hs},\ell}$ exports to the adversary handles which rely on *handle values* and internally uses handles which are built on concrete *key values*.

From the 1st to the 2nd column, we inline the code of the O oracle of $\texttt{M-Pr-io}$ (highlighted in pink). From the 2nd to the 3rd column, we replace the handle computation of h_{out} so that it uses the handle h_2 instead of the key k_2. This

mapping is injective, since these k_2 keys are all dishonest and the $\texttt{Nkey}_{dh}$ package exposes each dishonest keys only once due to the ∞ mapping. Moreover, note that handles of honest and dishonest keys are disjoint and thus, this change does not lead to confusions and/or more/less collisions in $\texttt{Log}_{hs}$.

Due to the disjointness between the handles in $K_{hs,\ell}$ in the $\texttt{Key}_{hs,\ell}$ package and the handles in $K_{hs,\ell}$ in $\texttt{Rcmprxtr}_{hs,\ell}$, we now also move the code of the $\mathsf{SET}_{hs,\ell}$ query from $\texttt{Rcmprxtr}_{hs,\ell} \to \texttt{M-Pr-io}^{\mathsf{xtr}}$ into the $\texttt{Key}_{hs,\ell}$ package.

Since now, external handles (i.e., those used by the adversary) are equal to internal handles (i.e., those which constitute the indices of $K_{hs,\ell}$), the $\mathsf{GET}_{hs,\ell}$, UNQ_{hs} and Q_{hs} queries can be directly forwarded to $\texttt{Key}_{hs,\ell}$ and $\texttt{Log}_{hs}$, respectively.

Moreover, we perform several minor changes from column 2 to column 3. We move the line $(k_1, hon_1) \leftarrow \mathsf{GET}_{hs,\ell}$ upwards and change the computation of the honesty (you can verify the equivalence of the changes by going through the honesty truth table). Finally, we also rename several of the variables and remove redundant computation. In particular, note that the first line of the $\texttt{Key}_{hs,\ell}$ package asserts that $\mathsf{name}(h) = hs$ and $\mathsf{level}(h) = \ell$, subsuming the check that $i \in \{(hs, \ell)\}$. All changes are highlighted in pink.

From the 3rd to the 4th column, we essentially swap the order of the **if** $-$**else** branches by computing $k \leftarrow \mathsf{xtr}(k_1, k_2)$ first and then overwriting it by a random key if needed. The $\mathsf{SET}_{hs,\ell}(h, hon, k)$ query is thus also moved outside of the branch since both branches perform it. Finally, **if** hon_2 is replaced by **if** $1 \wedge hon_2$.

Inequality 11 We define reduction $\mathcal{R}_{\mathrm{as},\ell}^{\mathrm{rmprio,xtr}}$ with the property that for $b \in \{0, 1\}$

$$\texttt{Gxtr3}_{as,\ell}^{b}, \overset{\text{code}}{\equiv} \mathcal{R}_{\mathrm{as},\ell}^{\mathrm{rmprio,xtr}} \to \texttt{G-M-Pr-io}_{\{as\}}^{(hsalt,\ell),\mathsf{xtr},b,D,D} \tag{41}$$

where we re-state $\texttt{Gxtr3}_{as,\ell}^{b}$ in Fig 49e for convenience. From there, it follows directly that for all adversaries $\mathcal{A}$,

$$\mathsf{Adv}(\mathcal{A}, \texttt{Gxtr1}_{as,\ell}^{b}) \leq \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{\mathrm{as},\ell}^{\mathrm{rmprio,xtr}}, \texttt{G-M-Pr-io}_{\{as\}}^{(hsalt,\ell),\mathsf{xtr},b,D,D})$$

It remains to provide the code equivalence proof of Equation 41 which is conceptually similar to the proof of Equation 39 except that the key is now the left input to xtr rather than the right input, and we have three handles in $\texttt{Nkey}_{0ikm}$ rather than one handle, see Figure 23b.

The game $\mathcal{R}_{\mathrm{as},\ell}^{\mathrm{rmprio,xtr}} \to \texttt{G-M-Pr-io}^{b}$ is provided in Figure 49f with the code of reduction $\mathcal{R}_{\mathrm{as},\ell}^{\mathrm{rmprio,xtr}}$ provided in Figure 53. As in the previous proofs, reduction $\mathcal{R}_{\mathrm{as},\ell}^{\mathrm{rmprio,xtr}}$ exports to the adversary handles which rely on handle values and internally relies on handles which rely on concrete key values. We prove Equation 39 by inlining $\texttt{M-Pr-io}^{\mathsf{xtr}}$ into $\mathcal{R}_{\mathrm{as},\ell}^{\mathrm{rmprio,xtr}}$ from the 1st to the 2nd column of Figure 53, highlighted in pink. From the 2nd to third column, we now use $\mathsf{xtr}\langle as, h_1, h_2\rangle$ as handle for the $\mathsf{SET}_{as,\ell}(h, hon, k)$ query instead of $\mathsf{xtr}\langle as, h_1, k_2\rangle$ which is a bijective mapping since $\texttt{Nkey}_{0ikm}$ three handles for distict key values (see Figure 23b). Additionally, we also remove the $\mathsf{OutIndex}$ computation and the assert that its result be in $\{(as, \ell)\}$. This is justified, since the first line

of $\mathsf{SET}_{as,\ell}(h, hon, k)$ asserts anyway that $\mathsf{name}(h) = as$ and $\mathsf{level}(h) = \ell$. Finally, since internally and externally, the same handles are used now, the queries $\mathsf{GET}_{as,\ell}$, Q_{as} and UNQ_{as} are directly forwarded. This concludes the proof of Equation 41.

E.4 Pre-image resistance

Similarly to pseudorandomness, we now relate the pre-image resistance advantages to the generic pseudorandomness game $\texttt{G-M-Pr-io}^{in,\mathsf{f},b,coll_{\mathrm{in}},coll_{\mathrm{Out}}}_{Out}$ Game. Before turning to the proof of Lemma E.9, we state and prove two helper lemmas. Lemma E.7 relates pseudorandomness and pre-image resistance, and Lemma E.8 relates games using the $\texttt{Xpd}_{n,0}$ package to $\texttt{G-M-Pr-io}^{in,\mathsf{f},b,coll_{\mathrm{in}},coll_{\mathrm{Out}}}_{Out}$.

Lemma E.7 (Pre-image-resistance to Pseudorandomness). *Consider the game* $\texttt{G-M-Pr-io}^{in,\mathsf{xpd},1,D,\mathbf{P}}_{Out}$*, where* $\mathbf{P}$ *is either* $\mathbf{D}$ *or* $\mathbf{Q}$*, where* $\mathbf{Q}$ *is either* $\mathbf{D}||R_{esalt}$ *or* $\mathbf{D}_{Out\setminus O^*}||\mathbf{F}_{Out\cap O^*}$*, both of which are defined below Inequality 42. For all adversaries* $\mathcal{A}$*, the following holds:*

$$\begin{aligned}
&\mathsf{Adv}(\mathcal{A} \to \mathcal{R}, \texttt{G-M-Pr-io}^{in,\mathsf{xpd},1,D,\mathbf{D}}_{Out}, \texttt{G-M-Pr-io}^{in,\mathsf{xpd},1,D,\mathbf{Q}}_{Out}) \\
\leq \sum_{alg\in\mathcal{H}} 2 \cdot \Bigg[& n_{alg} \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\mathsf{xpd}\text{-}alg,F,\mathbf{D}/\mathbf{Q}}_{\texttt{G-M-Pr-io}}, \texttt{Gpr}^{\mathsf{xpd}\text{-}alg,b}) \qquad (42) \\
&+ \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{cr,\mathsf{xpd},\mathbf{D}/\mathbf{P}} \to \mathcal{R}_{alg,\mathsf{xpd}}, \texttt{Gcr}^{\mathsf{xpd}\text{-}alg,b}) \\
&+ \frac{n^2_{alg,hon=1}}{2^{\mathsf{len}(alg)}} + \frac{n_{alg,hon=0} \cdot n_{alg,hon=1}}{2^{\mathsf{len}(alg)}} + (c+1) \cdot \frac{n^2_{alg,hon=1}}{2^{\mathsf{len}(alg)}} \Bigg],
\end{aligned}$$

where $n_{alg,hon=1}$ *is the number of* $\mathsf{SET}_{n_1,*}(h, 1, *)$ *queries which the adversary makes for handles* h *with* $\mathsf{alg}(h) = alg$*,* $n_{alg,hon=0}$ *is the number of* $\mathsf{SET}_{n_1,*}(h, 0, *)$ *queries which the adversary makes for handles* h *with* $\mathsf{alg}(h) = alg$*,* $n_{alg} = \max\{n_{alg,hon=1}, n_{alg,hon=0}\}$ *reduction* $\mathcal{R}^{cr,xtr,\mathbf{P}}$ *is defined in Figure 44, and reduction* $\mathcal{R}^{cr,xtr,\mathbf{D}/\mathbf{P}}$ *samples* $\mathcal{R}^{cr,xtr,\mathbf{D}}$ *or* $\mathcal{R}^{cr,xtr,\mathbf{P}}$ *uniformly at random, and reduction* $\mathcal{R}^{\mathsf{xpd}\text{-}alg,F,\mathbf{P}}_{\texttt{G-M-Pr-io}}$ *is defined in Lemma E.5, and* $\mathcal{R}^{\mathsf{xpd}\text{-}alg,F,\mathbf{D}/\mathbf{Q}}_{\texttt{G-M-Pr-io}}$ *samples one out of the reductions* $\mathcal{R}^{\mathsf{xpd}\text{-}alg,F,\mathbf{D}}_{\texttt{G-M-Pr-io}}$ *and* $\mathcal{R}^{\mathsf{xpd}\text{-}alg,F,\mathbf{Q}}_{\texttt{G-M-Pr-io}}$ *uniformly at random.*

$$\mathbf{D}||R_{esalt} : Out \to \{D, R\} \qquad n \mapsto \begin{cases} R, & \text{if } n = esalt \\ D & \text{otherwise} \end{cases}$$

$$\mathbf{D}_{Out\setminus O^*}||\mathbf{F}_{Out\cap O^*} : Out \to \{D, F\} \qquad n \mapsto \begin{cases} F, & \text{if } n \in O^* \\ D & \text{otherwise} \end{cases}$$

Proof. Inequality 42 encodes several reductions into one by sampling one out of two reductions (in order to obtain a concise inequality) and relies on reductions from Lemma A.2 and Lemma E.5. We start by reformulating Inequality 42 in

the following more intuitive way:

$$\begin{aligned}
&\mathsf{Adv}(\mathcal{A}, \texttt{G-M-Pr-io}_{Out}^{in,\mathsf{xpd},1,D,\mathbf{D}}, \texttt{G-M-Pr-io}_{Out}^{in,\mathsf{xpd},1,D,\mathbf{Q}}) \\
\leq &\mathsf{Adv}(\mathcal{A}, \texttt{G-M-Pr-io}_{Out}^{in,\mathsf{xpd},1,F,\mathbf{D}}, \texttt{G-M-Pr-io}_{Out}^{in,\mathsf{xpd},0,F,\mathbf{D}}) \\
+ &\mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\mathrm{cr},\mathsf{xpd},\mathbf{D}}, \texttt{Gacr}^{0,\mathsf{xpd}}, \texttt{Gacr}^{1,\mathsf{xpd}}) \\
+ &\mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\mathrm{cr},\mathsf{xpd},\mathbf{Q}}, \texttt{Gacr}^{0,\mathsf{xpd}}, \texttt{Gacr}^{1,\mathsf{xpd}}) \\
+ &\mathsf{Adv}(\mathcal{A}, \texttt{G-M-Pr-io}^{in,\mathsf{xpd},1,F,\mathbf{Q}}, \texttt{G-M-Pr-io}_{Out}^{in,\mathsf{xpd},0,F,\mathbf{Q}}) \\
+ &\sum_{alg \in \mathcal{H}} 2 \cdot \Big(\frac{n_{alg,hon=1}^2}{2^{\mathsf{len}(alg)}} + \frac{n_{alg,hon=0} \cdot n_{alg,hon=1}}{2^{\mathsf{len}(alg)}} \Big),
\end{aligned} \tag{43}$$

We derive Inequality 42 from Inequality 43 by upper-bounding (43) as follows:

$$\begin{aligned}
\sum_{alg \in \mathcal{H}} \Big(& n_{alg,hon=1} \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{\texttt{G-M-Pr-io}}^{\mathsf{xpd}\text{-}alg,F,\mathbf{D}}, \texttt{Gpr}^{\mathsf{xpd}\text{-}alg,b}) \\
&+ \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\mathrm{cr},\mathsf{xpd},\mathbf{D}} \to \mathcal{R}_{alg,\mathsf{xpd}}, \texttt{Gcr}^{\mathsf{xpd}\text{-}alg,b}) \\
&+ \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\mathrm{cr},\mathsf{xpd},\mathbf{Q}} \to \mathcal{R}_{alg,\mathsf{xpd}}, \texttt{Gcr}^{\mathsf{xpd}\text{-}alg,b}) \\
&+ n_{alg,hon=1} \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{\texttt{G-M-Pr-io}}^{\mathsf{xpd}\text{-}alg,F,\mathbf{Q}}, \texttt{Gpr}^{\mathsf{xpd}\text{-}alg,b}) \\
&+ 2 \cdot \Big(\frac{n_{alg,hon=1}^2}{2^{\mathsf{len}(alg)}} + \frac{n_{alg,hon=0} \cdot n_{alg,hon=1}}{2^{\mathsf{len}(alg)}} + (c+1) \cdot \frac{n_{alg,hon=1}^2}{2^{\mathsf{len}(alg)}} \Big) \Big),
\end{aligned} \tag{44}$$

by replacing

$$\begin{aligned}
&\mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\mathrm{cr},\mathsf{xpd},\mathbf{P}}, \texttt{Gacr}^{0,\mathsf{xpd}}, \texttt{Gacr}^{1,\mathsf{xpd}}) \text{ by} \\
&\sum_{alg \in \mathcal{H}} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\mathrm{cr},\mathsf{xpd},\mathbf{P}} \to \mathcal{R}_{alg,\mathsf{xpd}}, \texttt{Gcr}^{\mathsf{xpd}\text{-}alg,b})
\end{aligned}$$

for $\mathbf{P} \in \{\mathbf{D}, \mathbf{Q}\}$ using Lemma A.2 and replacing

$$\begin{aligned}
&\mathsf{Adv}(\mathcal{A}, \texttt{G-M-Pr-io}^{in,\mathsf{xpd},1,F,\mathbf{P}}, \texttt{G-M-Pr-io}_{Out}^{in,\mathsf{xpd},0,F,\mathbf{P}}) \text{ by} \\
&\sum_{alg \in \mathcal{H}} n_{alg,hon=0} \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{\texttt{G-M-Pr-io}}^{\mathsf{xpd}\text{-}alg,F,\mathbf{P}}, \texttt{Gpr}^{\mathsf{xpd}\text{-}alg,b}) + (c+1) \cdot \frac{n_{alg,hon=1}^2}{2^{\mathsf{len}(alg)}}
\end{aligned}$$

using Lemma E.5. From (44), we then obtain Inequality 42 by encoding the two collision-resistance reductions into a single one and by encoding the two pseudo-randomness reductions into a single one. It thus remains to prove Inequality 43.

The proof of Inequality 43 proceeds by a sequence of game-hops. In the first game hop, the input `Log` patterns is switched from a D to an F pattern:

$$\begin{aligned}
&\Pr\Big[1 = \mathcal{A} \to \texttt{G-M-Pr-io}_{Out}^{in,\mathsf{xpd},1,D,\mathbf{D}}\Big] \\
\leq &\Pr\Big[1 = \mathcal{A} \to \texttt{G-M-Pr-io}_{Out}^{in,\mathsf{xpd},1,F,\mathbf{D}}\Big] + \\
&\qquad \frac{n_{alg,hon=1}^2}{2^{\mathsf{len}(alg)}} + \frac{n_{alg,hon=0} \cdot n_{alg,hon=1}}{2^{\mathsf{len}(alg)}}
\end{aligned} \tag{45}$$

To see that Inequality 45 holds, first note that the adversary yields no information about the honest keys stored in $\texttt{Key}^1_{in}$, since (a) they are sampled uniformly at random and (b) the output $\texttt{Key}^1_n$ packages overwrite concretely derived values by random ones. Thus, the adversary does not see any dependent operations on the input keys, unless the F pattern on $\texttt{Log}_{in}$ aborts. The adversary creates up to $n_{alg,hon=0}$ dishonest keys for algorithm alg, each of which has a $\frac{n_{alg,hon=1}}{2^{\mathsf{len}(alg)}}$ chance of agreeing with an honest key. Thus, we lose an additive term of $\frac{n_{alg,hon=0} \cdot n_{alg,hon=1}}{2^{\mathsf{len}(alg)}}$. In addition, since the F pattern also aborts on collisions between two honest keys, we also lose a birthday bound in the number of honest keys, i.e., $\frac{n^2_{alg,hon=1}}{2^{\mathsf{len}(alg)}}$. We now apply the pseudorandomness assumption and thus, by definition and triangle inequality, we obtain:

$$\begin{aligned}
&\Pr\left[1 = \mathcal{A} \to \texttt{G-M-Pr-io}^{in,\mathsf{xpd},1,F,\mathbf{D}}_{Out}\right] \\
\leq &\Pr\left[1 = \mathcal{A} \to \texttt{G-M-Pr-io}^{in,\mathsf{xpd},0,F,\mathbf{D}}_{Out}\right] + \\
&\mathsf{Adv}(\mathcal{A}, \texttt{G-M-Pr-io}^{in,\mathsf{xpd},1,F,\mathbf{D}}_{Out}, \texttt{G-M-Pr-io}^{in,\mathsf{xpd},0,F,\mathbf{D}}_{Out})
\end{aligned}$$

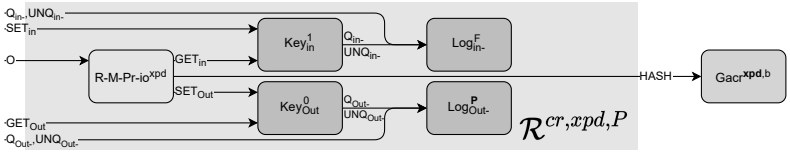


Fig. 44: Collision Resistance Reduction for `G-M-Pr-io` where P is either $\mathbf{D}$, $\mathbf{F}_{Out \cap O^*} || \mathbf{D}_{Out \setminus O^*}$ or $\mathbf{R}_{esalt} || \mathbf{D}_{CN \setminus \{esalt\}}$.

Next, we use collision-resistance of xpd. We decompose the `M-Pr-io` package into reduction $\texttt{R-M-Pr-io}^{\mathsf{xpd}}$ (Fig. 44) and game $\texttt{Gcr}^{0,\mathsf{xpd}}$ and then apply the collision-resistance assumption.

$$\begin{aligned}
&\Pr\left[1 = \mathcal{A} \to \texttt{G-M-Pr-io}^{in,\mathsf{xpd},0,F,\mathbf{D}}_{Out}\right] \\
= &\Pr\left[1 = \mathcal{A} \to \mathcal{R}^{cr,xpd,\mathbf{D}} \to \texttt{Gcr}^{0,\mathsf{xpd}}\right] \quad (46) \\
\leq &\Pr\left[1 = \mathcal{A} \to \mathcal{R}^{cr,xpd,\mathbf{D}} \to \texttt{Gcr}^{1,\mathsf{xpd}}\right] + \\
&\mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{cr,xpd,\mathbf{D}}, \texttt{Gcr}^{0,\mathsf{xpd}}, \texttt{Gcr}^{1,\mathsf{xpd}})
\end{aligned}$$

$\underline{\texttt{R-M-Pr-io}^{\mathsf{xpd}}}$
$\mathsf{O}^{\mathsf{xpd}}(h, args)$

$(k, hon) \leftarrow \mathsf{GET}_{in}(h)$
$k_{out} \leftarrow \mathsf{HASH}(k, args)$
$h_{out} \leftarrow \mathsf{inj}\langle h, args\rangle$
$i \leftarrow \mathsf{OutIndex}(h_{out})$
assert $i \in Out$
return $\mathsf{SET}_i(h_{out}, hon, k_{out})$

Fig. 43: Code of $\texttt{R-M-Pr-io}^{\mathsf{xpd}}$

We now observe that we can replace the pattern $\mathbf{D}$ on the output keys by the pattern $\mathbf{F}_A || \mathbf{D}_B$ in a perfect equivalence step, analogously to the proof of Lemma D.2. Since the input keys are unique and $\texttt{Gcr}^{1,\mathsf{xpd}}$ aborts on colliding derived keys, the F patterns on the output are not triggered. Subsequently, we de-idealize collision-resistance:

$$\begin{aligned}
&\Pr\left[1 = \mathcal{A} \to \mathcal{R}^{cr,xpd,\mathbf{D}} \to \mathtt{Gcr}^{1,\mathsf{xpd}}\right] \\
=&\Pr\left[1 = \mathcal{A} \to \mathcal{R}^{cr,xpd,\mathbf{Q}} \to \mathtt{Gcr}^{1,\mathsf{xpd}}\right] \\
\leq&\Pr\left[1 = \mathcal{A} \to \mathcal{R}^{cr,xpd,\mathbf{Q}} \to \mathtt{Gcr}^{0,\mathsf{xpd}}\right] + \\
&\mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{cr,xpd,\mathbf{Q}}, \mathtt{Gcr}^{1,\mathsf{xpd}}, \mathtt{Gcr}^{0,\mathsf{xpd}})
\end{aligned}$$

The next proof step is the inverse analogue of Equality 46. I.e., we recompose reduction $\mathtt{R\text{-}M\text{-}Pr\text{-}io}^{\mathsf{xpd}}$ and game $\mathtt{Gcr}^{0,\mathsf{xpd}}$ into game $\mathtt{M\text{-}Pr\text{-}io}$. Afterwards, we apply the pseudorandomness assumption.

$$\begin{aligned}
&\Pr\left[1 = \mathcal{A} \to \mathcal{R}^{cr,xpd,\mathbf{Q}} \to \mathtt{Gcr}^{0,\mathsf{xpd}}\right] \\
=&\Pr\left[1 = \mathcal{A} \to \mathtt{G\text{-}M\text{-}Pr\text{-}io}_{Out}^{in,\mathsf{xpd},0,F,\mathbf{Q}}\right] \\
\leq&\Pr\left[1 = \mathcal{A} \to \mathtt{G\text{-}M\text{-}Pr\text{-}io}_{Out}^{in,\mathsf{xpd},0,F,\mathbf{Q}}\right] + \\
&\mathsf{Adv}(\mathcal{A}, \mathtt{G\text{-}M\text{-}Pr\text{-}io}_{Out}^{in,\mathsf{xpd},1,F,\mathbf{Q}}, \mathtt{G\text{-}M\text{-}Pr\text{-}io}_{Out}^{in,\mathsf{xpd},0,F,\mathbf{Q}})
\end{aligned}$$

Finally, we perform the analogous (inverse) transformation of Inequality 45 and replace the F pattern on the input keys by a D pattern. We note, once again that the adversary has no information on the input keys and thus, we lose a random guessing probability as well as collisions between honest keys.

$$\begin{aligned}
&\Pr\left[1 = \mathcal{A} \to \mathtt{G\text{-}M\text{-}Pr\text{-}io}_{Out}^{in,\mathsf{xpd},1,F,\mathbf{Q}}\right] \\
\leq&\Pr\left[1 = \mathcal{A} \to \mathtt{G\text{-}M\text{-}Pr\text{-}io}_{Out}^{in,\mathsf{xpd},1,D,\mathbf{Q}}\right] + \\
&\qquad\qquad \frac{n^2_{alg,hon=1}}{2^{\mathsf{len}(alg)}} + \frac{n_{alg,hon=0} \cdot n_{alg,hon=1}}{2^{\mathsf{len}(alg)}}
\end{aligned}$$

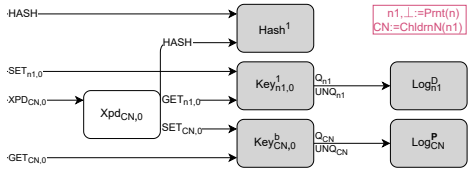
Lemma E.7 then follows from the triangle inequality.

We can now turn to showing that games which use the $\mathtt{Xpd}_{n,0}$ package can be closely related to $\mathtt{G\text{-}M\text{-}Pr\text{-}io}_{Out}^{in,\mathsf{f},b,coll_{\mathrm{in}},coll_{\mathrm{Out}}}$. The proof of this statement will be closely related to the proof of Equation 36.

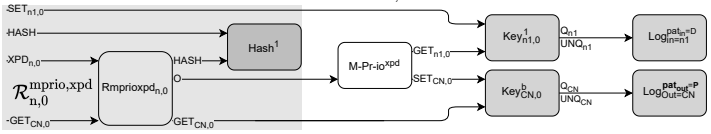
Lemma E.8 (Generalized Equation (36)). *Let n_1 be a name and let $CN := \mathsf{ChldrnN}(n_1)$ denote the children of n_1. Furthermore, let $\mathbf{P}$ be an arbitrary pattern $\mathbf{P} : CN \to \{D, F, R\}$. Then, for all $b \in \{0, 1\}$,*

$$\mathtt{Gxpd}_{n,0}^{b,\mathbf{P}}, \overset{code}{\equiv} \mathcal{R}_{n,0}^{rmprio,\mathsf{xpd}} \to \mathtt{G\text{-}M\text{-}Pr\text{-}io}_{CN,0}^{(n_1,0),\mathsf{xpd},b,D,\mathbf{P}}, \tag{47}$$

where $\mathtt{Gxpd}_{n,0}^{b,\mathbf{P}}$ is defined in Figure 45a and $\mathcal{R}_{n,0}^{rmprio,\mathsf{xpd}}$ is defined in Figure 50b.



(a) $\texttt{Gxpd}^{b,\mathbf{P}}_{n,0}$



(b) Game $\mathcal{R}^{\text{rmprio,xpd}}_{\text{n},0} \to \texttt{G-M-Pr-io}^{(n_1,0),\text{xpd},b,D,D_{CN}}_{CN,0}$ where $\texttt{Rmprioxpd}_{CN,0}$ is defined in Fig. 54

Fig. 45: Reductions for Inequality 47

Proof. We include the graph of $\mathcal{R}^{\text{rmprio,xpd}}_{\text{n},0} \to \texttt{G-M-Pr-io}^{(n_1,0),\text{xpd},b,D,D_{CN}}_{CN,0}$ in Figure 45b. The proof of Equation 47 proceeds by inlining the code of `M-Pr-io` into $\texttt{Rmprioxpd}_{CN,0}$, see Figure 54. We also used Figure 54 in the proof of Equation 36, and indeed, all arguments apply here, since Equation 47 and Equation 36 differ only in the pattern on the output $\texttt{Log}^{\mathbf{P}}_{CN}$.

With Lemma E.7 and Lemma E.8 at hand, we can now re-state and prove Lemma E.9 (Pre-Image Resistance) which relates our two pre-image resistance assumptions (Figure 24i and Figure 24h) directly to monolithic, non-agile pseudo-randomness and non-agile collision-resistance.

$\texttt{LevelMap}^{d}_{In,Evl,Out}$

State	Parameter	
M level map	In	input names
	Evl	evaluation names
	Out	output names
	d	level count

$\mathsf{SET}_{n\in In,0..d}(h, hon, k)$

assert $\mathsf{level}(h) = \ell$
assert $\mathsf{name}(h) = n$
$alg \leftarrow \mathsf{Alg}(h)$
if $M_{n,\ell}[h] = \perp$:
  $M_{n,\ell}[h] \leftarrow \mathsf{NewHandle}(alg, n, M_{0..d}, 0)$
return $\mathsf{SET}_{n,0}(M_{n,\ell}[h], hon, k)$

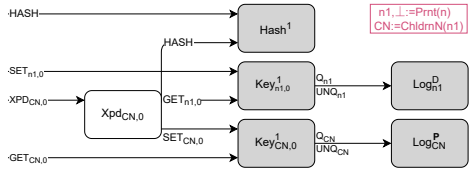
$\mathsf{XPD}_{n\in Evl,0..d}(h_1, r, args)$

$n_1, \_ \leftarrow \mathsf{PrntN}(n)$
assert $M_{n_1,\ell}[h_1] \neq \perp$
$label \leftarrow \mathsf{Labels}(n, r)$
$h \leftarrow \mathsf{xpd}\langle n, label, h_1, args\rangle$
$M_{n,\ell}[h] \leftarrow \mathsf{XPD}_{n,0}(M_{n_1,\ell}[h_1], args)$
return $M_{n,\ell}[h]$

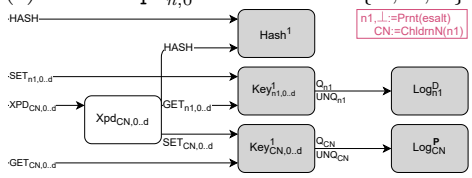
$\mathsf{GET}_{n\in Out,0..d}(h)$

assert $M_{n,\ell}[h] \neq \perp$
return $\mathsf{GET}_{n,0}(M_{n,\ell}[h])$

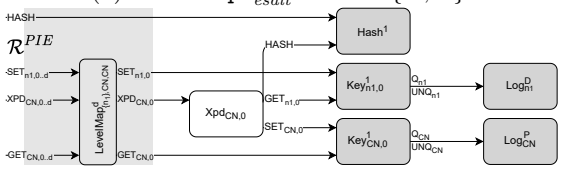
Fig. 46: Code of injective level mapping `LevelMap`



(a) Game $\texttt{Gxpd}^{1,\mathbf{P}}_{n,0}$ for $\mathbf{P} : CN \to \{R, D, F\}$



(b) Game $\texttt{Gxpd}^{\mathrm{PI},\mathbf{P}}_{esalt}$ for $P \in \{D, R\}$



(c) Game $\mathcal{R}^{\mathrm{mprio},\mathsf{xpd}}_{n,\ell} \to \texttt{G-M-Pr-io}^{(n_1,\ell),\mathsf{xpd},b,D,D_{CN}}_{CN,0}$ for $b \in \{0,1\}$, where $(n_1, \_) = \mathsf{PrntN}(n)$ and $CN = \mathsf{ChldrnN}(n_1)$. Reduction $\mathcal{R}^{\mathrm{mprio},\mathsf{xpd}}_{n,\ell}$ in grey.

Fig. 47: Reductions for Inequality 12

Lemma E.9. *Let $\mathcal{A}$ be an adversary that generates at most $t^{hon=1}_{es,alg}$ honest and $t^{hon=0}_{es,alg}$ dishonest es keys, and let $t_{es,alg}$ be $\max\{t^{hon=0}_{es,alg}, t^{hon=1}_{es,alg}\}$. We have*

$$\mathsf{Adv}(\mathcal{A}, \texttt{Gpi}^{R}_{esalt}, \texttt{Gpi}^{D}_{esalt}) \tag{12}$$

$$\leq \sum_{alg \in \mathcal{H}} 2 \cdot \Bigg[s_{esalt,alg} \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{alg}_{esalt,pi}, \texttt{Gpr}^{\mathsf{xpd}\text{-}alg,b}) + \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{alg}_{esalt,pi\text{-}cr}, \texttt{Gcr}^{\mathsf{hash}\text{-}alg,b}) + (c+3) \cdot \frac{t^2_{es,alg}}{2^{\mathsf{len}(alg)}} \Bigg],$$

where $\mathcal{R}^{alg}_{esalt,pi} := \mathcal{R}^{PIE} \to \mathcal{R}^{mprio,xpd}_{n,0} \to \mathcal{R}^{\mathsf{xpd}\text{-}alg,D,\mathbf{D}/\mathbf{R}}_{\texttt{G-M-Pr-io}}$ *and*

$\mathcal{R}^{alg}_{esalt,pi\text{-}cr} := \mathcal{R}^{PIE} \to \mathcal{R}^{mprio,xpd}_{n,0} \to \mathcal{R}^{cr,\mathsf{xpd},\mathbf{D}/\mathbf{R}} \to \mathcal{R}_{alg,\mathsf{xpd}}$,

c is a small constant which depends on the min-entropy of xpd *on random inputs and $\mathbf{R}$ is an abbreviation for $\mathbf{R} : \mathsf{ChldrnN}(es) \to \{R, D\}$ being defined as R on esalt and D everywhere else.*

Let the parent set of O^* be defined as $PO^* := \{n_1 : \exists n \in O^* : (n_1, \_) = \mathsf{PrntN}(n)\}$ and the sibling set of O^* as $SO^* := \bigcup_{n_1 \in PO^*} \mathsf{ChldrnN}(n_1)$. Let $\mathcal{A}$ be an adversary such that for each n in the representative set $SO^* \cap XPR$, $\mathcal{A}$

generates at most $s_{n,alg} = t_{n,alg}^{hon=1}$ *honest and* $t_{n,alg}^{hon=0}$ *dishonest* n *keys for* n_1 *with* $(n_1, \_) = \mathsf{PrntN}(n)$*, and let* $t_{n,alg}$ *be* $\max\{t_{n,alg}^{hon=0}, t_{n,alg}^{hon=1}\}$*. We have*

$$\mathsf{Adv}(\mathcal{A}, \mathtt{Gpi}_{O^*}^{D}, \mathtt{Gpi}_{O^*}^{F}) \tag{13}$$

$$\leq \sum_{alg \in \mathcal{H}, n \in SO^* \cap XPR} 2 \cdot \Bigg[s_{n,alg} \cdot \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{n,pi}^{alg}, \mathtt{Gpr}^{\mathsf{xpd}\text{-}alg,b}) + \mathsf{Adv}(\mathcal{A} \to \mathcal{R}_{n,pi\text{-}cr}^{alg}, \mathtt{Gcr}^{\mathsf{hash}\text{-}alg,b}) + (c+3) \cdot \frac{t_{n,alg}^2}{2^{\mathsf{len}(alg)}} \Bigg],$$

$$\textit{where} \quad \mathcal{R}_{n,pi}^{alg} := \mathcal{R}_{n}^{PIO} \to \mathcal{R}_{n,0}^{mprio,xpd} \to \mathcal{R}_{\mathtt{G\text{-}M\text{-}Pr\text{-}io}}^{\mathsf{xpd}\text{-}alg,D,\mathbf{D}/\mathbf{F}} \textit{ and}$$
$$\mathcal{R}_{n,pi\text{-}cr}^{alg} := \mathcal{R}_{n}^{PIO} \to \mathcal{R}_{n,0}^{mprio,xpd} \to \mathcal{R}^{cr,\mathsf{xpd},\mathbf{D}/\mathbf{F}} \to \mathcal{R}_{alg,\mathsf{xpd}},$$

c is a small constant which depends on the min-entropy of xpd *on random inputs, and* $\mathbf{F} : \mathsf{ChldrnN}(n_1) \to \{D, F\}$ *is F on the intersection of O^* and* $\mathsf{ChldrnN}(n_1)$ *and D, else.*

Proof. We first prove Inequality 12 and then Inequality 13. For Inequality 12, let $CN(es) := \mathsf{ChldrnN}(es)$. To obtain Inequality 12, we combine Inequality 42, Equality 47 and the following code-equivalence statements:

$$\begin{aligned} \mathtt{Gpi}_{esalt}^{D} &\overset{\text{func}}{\equiv} \mathcal{R}^{\text{PIE}} \to \mathtt{Gxpd}_{esalt,0}^{1,\mathbf{D}_{CN(es)}} \\ \mathtt{Gpi}_{esalt}^{R} &\overset{\text{func}}{\equiv} \mathcal{R}^{\text{PIE}} \to \mathtt{Gxpd}_{esalt,0}^{1,\mathbf{D}_{CN(es)\setminus\{esalt\}} || \mathbf{R}_{esalt}} \end{aligned} \tag{48}$$

where $\mathcal{R}^{\text{PIE}}$ is defined in Figure 47c. The reduction $\mathcal{R}^{\text{PIE}}$ performs an injective handle mapping, i.e., the adversary uses handles of any level whereas internally, $\mathcal{R}^{\text{PIE}}$ only makes calls with handles h such that $\mathsf{name}(h) = es$ and $\mathsf{level}(h) = 0$. See Figure 46 for the code of $\mathtt{LevelMap}_{\{n_1\},CN,CN}^{d}$ which is the main building block of $\mathcal{R}^{\text{PIE}}$.

The functional equivalence between game $\mathtt{G\text{-}M\text{-}Pr\text{-}io}_{CN(es),0}^{es,\mathsf{xpd},1,D,P}$ and $\mathcal{R}^{\text{PIE}} \to \mathtt{Gxpd}_{es,0}^{b}$ follows from the injectivity of the handle mapping, i.e., both games consume the same number of random strings (of the same length) and $\mathcal{R}^{\text{PIE}}$ maps these consistently. This concludes the proof of Equation 48.

We now apply Equality 48, Equality 47 and Inequality 42 successively to obtain Inequality 12. We introduce the abbreviation DD for $\mathbf{D}_{CN(es)}$ and DR

for $\mathbf{D}_{CN(es)\setminus\{esalt\}}||\mathbf{R}_{esalt}$ for sake of conciseness in the 3rd equality below:

$$
\begin{aligned}
&\mathsf{Adv}(\mathcal{A}, \mathtt{Gpi}^{R}_{esalt}, \mathtt{Gpi}^{D}_{esalt}) \\
\overset{(48)}{=}&\mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\mathrm{PIE}}, \mathtt{Gxpd}^{1,\mathbf{D}_{CN(es)}}_{esalt,0}, \mathtt{Gxpd}^{1,\mathbf{D}_{CN(es)\setminus\{esalt\}}||\mathbf{R}_{esalt}}_{esalt,0}) \\
\overset{(47)}{=}&\mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\mathrm{PIE}} \to \mathcal{R}^{\mathrm{rmprio},\mathsf{xpd}}_{esalt,0}, \mathtt{G\text{-}M\text{-}Pr\text{-}io}^{(n_1,0),\mathsf{xpd},1,D,\mathbf{D}_{CN(es)}}_{CN(es),0}, \\
&\qquad\qquad \mathtt{G\text{-}M\text{-}Pr\text{-}io}^{(n_1,0),\mathsf{xpd},1,D,\mathbf{D}_{CN(es)\setminus\{esalt\}}||\mathbf{R}_{esalt}}_{CN(es),0}) \\
=&\mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\mathrm{PIE}} \to \mathcal{R}^{\mathrm{rmprio},\mathsf{xpd}}_{esalt,0}, \mathtt{G\text{-}M\text{-}Pr\text{-}io}^{(n_1,0),\mathsf{xpd},1,D,DD}_{CN(es),0}, \\
&\qquad\qquad \mathtt{G\text{-}M\text{-}Pr\text{-}io}^{(n_1,0),\mathsf{xpd},1,D,DR}_{CN(es),0}) \\
\overset{\text{Lem. E.7}}{\leq}&\sum_{alg\in\mathcal{H}} 2 \cdot \Bigg[n_{alg} \cdot \\
&\mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\mathrm{PIE}} \to \mathcal{R}^{\mathrm{rmprio},\mathsf{xpd}}_{esalt,0} \to \mathcal{R}^{\mathsf{xpd}\text{-}alg,D,DD/DR}_{\mathtt{G\text{-}M\text{-}Pr\text{-}io}}, \mathtt{Gpr}^{\mathsf{xpd}\text{-}alg,b}) \\
+&\mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\mathrm{PIE}} \to \mathcal{R}^{\mathrm{rmprio},\mathsf{xpd}}_{esalt,0} \to \mathcal{R}^{\mathrm{cr},\mathsf{xpd},DD/DR} \to \mathcal{R}_{alg}, \mathtt{Gcr}^{\mathsf{xpd}\text{-}alg,b}) \\
+&\Big(\frac{n^2_{alg,hon=1}}{2^{\mathsf{len}(alg)}} + \frac{n_{alg,hon=0} \cdot n_{alg,hon=1}}{2^{\mathsf{len}(alg)}} + (c+1) \cdot \frac{n^2_{alg}}{2^{\mathsf{len}(alg)}} \Big) \Bigg],
\end{aligned}
$$

which concludes the proof of Inequality 12.

The proof of Inequality 13 proceeds via a hybrid argument over the different input names in PO^*, i.e., we first slice the game to the different sub-games keyed only by a single name $n \in \mathrm{PO}^*$. While slicing the games according to $n \in \mathrm{PO}^*$, we also apply an injective handle mapping which maps all handles to level 0 handles in order to avoid the use of multiple input key packages We now first state Lemma E.10 (slicing), then prove Inequality 13, and then prove Lemma E.10 (slicing).

Lemma E.10 (Slicing). *Let $<$ be some arbitrary complete order on $SO^* \cap XPR$. Then, for all adversaries $\mathcal{A}$, we have*

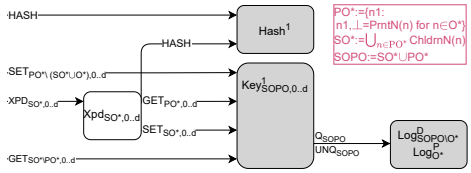
$$
\mathsf{Adv}(\mathcal{A}, \mathtt{Gpi}^{D}_{O^*}, \mathtt{Gpi}^{F}_{O^*}) \leq \sum_{n\in SO^*\cap XPR} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{PIO}_{n}, \mathtt{Gxpd}^{1,\mathbf{D}}_{n,0}, \mathtt{Gxpd}^{1,\mathbf{F}}_{n,0}),
$$

where $\mathcal{R}^{PIO}_{n_1}$ is defined in Figure 48b and Figure 46, and $\mathtt{Gxpd}^{b,\mathbf{F}}_{n,0}$ (with $P \in \{D,F\}$) is defined in Figure 47a.

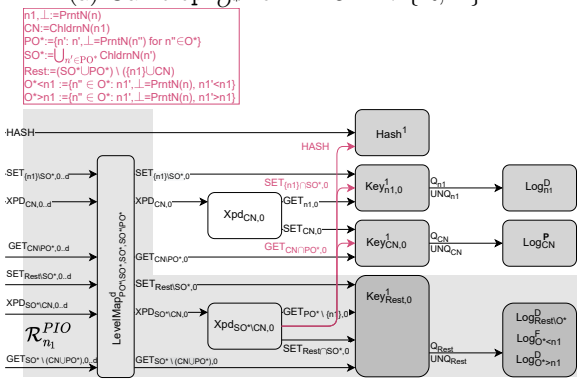
Using Lemma E.10 (slicing), we now prove Inequality 13. In the following sequence of inequalities, for $n \in SO^* \cap XPR$, we will often need n_1 such that $(n_1, \_) = \mathsf{PrntN}(n)$. In this case, we only write n_1 for conciseness of notation. Moreover, we write $CN(n_1)$ for $\mathsf{ChldrnN}(n_1)$ and $\mathbf{F}$ for $\mathbf{F}_{CN(n_1)\cap SO^*}||\mathbf{D}_{CN(n_1)\setminus SO^*}$.

$$
\begin{aligned}
&\mathsf{Adv}(\mathcal{A}, \mathtt{Gpi}^{D}_{O^*}, \mathtt{Gpi}^{F}_{O^*}) \\
\overset{\text{Lem. E.10}}{\leq} &\sum_{n \in \mathrm{SO}^* \cap XPR} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\mathrm{PIO}}_{n}, \mathtt{Gxpd}^{1,\mathbf{D}}_{n,0}, \mathtt{Gxpd}^{1,\mathbf{F}}_{n,0}) \\
\overset{(47)}{=} &\sum_{n \in \mathrm{SO}^* \cap XPR} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\mathrm{PIO}}_{n} \to \mathcal{R}^{\mathrm{rmprio},\mathsf{xpd}}_{n,0}, \mathtt{G\text{-}M\text{-}Pr\text{-}io}^{(n_1,0),\mathsf{xpd},1,D,\mathbf{D}}_{CN(n_1),0}, \\
&\qquad\qquad \mathtt{G\text{-}M\text{-}Pr\text{-}io}^{(n_1,0),\mathsf{xpd},1,D,\mathbf{F}}_{CN(n_1),0}) \\
\overset{\text{Lem. E.7}}{\leq} &\sum_{n \in \mathrm{SO}^* \cap XPR} \sum_{alg \in \mathcal{H}} 2 \cdot \Bigg[n_{alg} \cdot \\
&\qquad \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\mathrm{PIO}}_{n} \to \mathcal{R}^{\mathrm{rmprio},\mathsf{xpd}}_{n,0} \to \mathcal{R}^{\mathsf{xpd}\text{-}alg,D,\mathbf{D}/\mathbf{F}}_{\mathtt{G\text{-}M\text{-}Pr\text{-}io}}, \mathtt{Gpr}^{\mathsf{xpd}\text{-}alg,b}) \\
&\qquad + \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\mathrm{PIO}}_{n} \to \mathcal{R}^{\mathrm{rmprio},\mathsf{xpd}}_{n,0} \to \mathcal{R}^{\mathrm{cr},\mathsf{xpd},\mathbf{D}/\mathbf{F}} \to \mathcal{R}_{alg}, \mathtt{Gcr}^{\mathsf{xpd}\text{-}alg,b}) \\
&\qquad + \Big(\frac{n^2_{alg,hon=1}}{2^{\mathsf{len}(alg)}} + \frac{n_{alg,hon=0} \cdot n_{alg,hon=1}}{2^{\mathsf{len}(alg)}} + (c+1) \cdot \frac{n^2_{alg}}{2^{\mathsf{len}(alg)}} \Big) \Bigg],
\end{aligned}
$$

This concludes the proof of Inequality 13 except for the proof of Lemma E.10 (slicing), which we turn to now.



(a) Game $\mathtt{Gpi}^{\mathbf{P}}_{O^*}$ for $\mathbf{P} : CN \to \{R, D\}$



(b) Game $\mathtt{Hpio}_n := \mathcal{R}^{\mathrm{PIO}}_{n} \to \mathtt{Gxpd}^{1,\mathbf{D}}_{n,0}$ for $n \in O^* \cap XPR$. Reduction $\mathcal{R}^{\mathrm{PIO}}_{n}$ in grey.

Fig. 48: Reductions for Inequality 13

Proof of Lemma E.10. The proof proceeds via a hybrid argument over the ordering $<$ of $O^* \cap XPR$, denote by $n^{\min}$ the minimal element in $O^* \cap XPR$, by $n^{\max}$ the maximal element in $O^* \cap XPR$, by $n+1$ the successor of n in the ordering $<$, where we also add $n_1^{\max}+1$ to the ordering $<$ as additional new maximal element. Let $\mathcal{A}$ be an adversary, for $n \in \mathrm{SO}^* \cap XPR$, define $\mathtt{Hpio}_n$ as the game in Figure 48b, and define $\mathtt{Hpio}_{n^{\max}+1}$ as $\mathtt{Gpi}^F_{O^*}$. We need the following functional equivalences:

$$\mathtt{Gpi}^D_{O^*} \overset{\text{func}}{\equiv} \mathtt{Hpio}_{n^{\min}} \text{(first hybrid)} \tag{49}$$

$$\mathcal{R}^{\mathrm{PIO}}_n \to \mathtt{Gxpd}^{1,\mathbf{D}}_{n,0} \overset{\text{func}}{\equiv} \mathtt{Hpio}_n \tag{50}$$

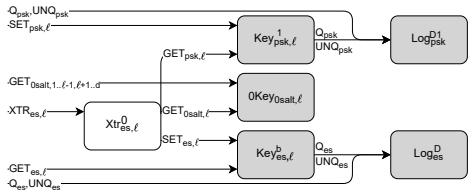
$$\mathcal{R}^{\mathrm{PIO}}_n \to \mathtt{Gxpd}^{1,\mathbf{F}_{CN(n_1)\cap SO^*}||\mathbf{D}_{CN(n_1)\setminus SO^*}}_{n,0} \overset{\text{func}}{\equiv} \mathtt{Hpio}_{n+1} \tag{51}$$

$$\mathtt{Gpi}^F_{O^*} \overset{\text{func}}{\equiv} \mathtt{Hpio}_{n^{\max}+1} \text{(last hybrid)} \tag{52}$$

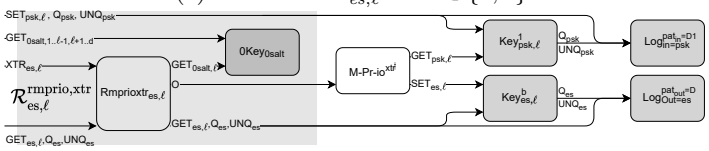
To show that (49)-(52) imply Lemma E.10, we provide the telescopic sum computation for completeness.

$$\begin{aligned}
&\mathsf{Adv}(\mathcal{A}, \mathtt{Gpi}^D_{O^*} \overset{\text{func}}{,} \mathtt{Gpi}^F_{O^*}) \\
\overset{(49)+(52)}{=}\ &\mathsf{Adv}(\mathcal{A}, \mathtt{Hpio}_{n^{\min}}, \mathtt{Hpio}_{n^{\max}+1}) \\
\overset{\text{tel. sum}}{\leq}\ &\sum_{n \in \mathrm{SO}^* \cap XPR} \mathsf{Adv}(\mathcal{A}, \mathtt{Hpio}_n, \mathtt{Hpio}_{n+1}) \\
\overset{(50)+(51)}{\leq}\ &\sum_{n \in \mathrm{SO}^* \cap XPR} \mathsf{Adv}(\mathcal{A}, \mathcal{R}^{\mathrm{PIO}}_n \to \mathtt{Gxpd}^{1,\mathbf{D}}_{n,0}, \mathcal{R}^{\mathrm{PIO}}_n \to \mathtt{Gxpd}^{1,\mathbf{F}}_{n,0}) \\
\leq\ &\sum_{n \in \mathrm{SO}^* \cap XPR} \mathsf{Adv}(\mathcal{A} \to \mathcal{R}^{\mathrm{PIO}}_n, \mathtt{Gxpd}^{1,\mathbf{D}}_{n,0}, \mathtt{Gxpd}^{1,\mathbf{F}}_{n,0})
\end{aligned}$$

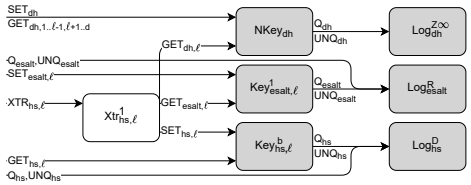
It remains to show Equations (49)-(52) are true. (52) and (50) hold by definition. (51) can be verified by re-ordering the packages in Fig. 48b (or equivalently, by taking the unions over the indices). For (49), observe that if $n = n^{\min}$, then all patterns in the lower `Log` packages in Fig. 48b have a D pattern, and thus we obtain $\mathtt{Gpi}^D_{O^*}$. This concludes the proof of Lemma E.10.



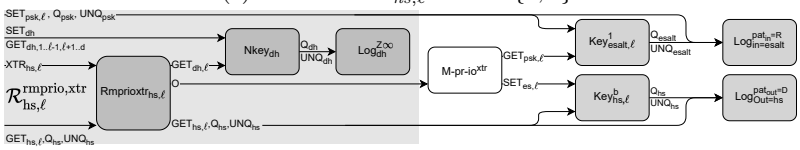
(a) Game $\texttt{Gxtr1}^b_{es,\ell}$ for $b \in \{0,1\}$



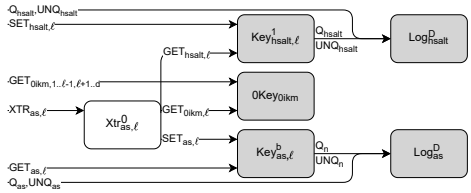
(b) Game $\mathcal{R}^{rmprio,\mathsf{xtr}}_{es,\ell} \to \texttt{G-M-Pr-io}^{(psk,\ell),\mathsf{xtr}^\dagger,b,D1,D}_{\{(es,\ell)\}}$, $b \in \{0,1\}$, reduction $\mathcal{R}^{rmprio,\mathsf{xtr}}_{es,\ell}$ in grey.



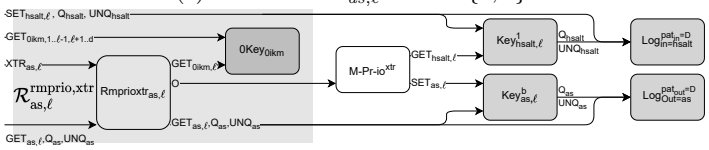
(c) Game $\texttt{Gxtr2}^b_{hs,\ell}$ for $b \in \{0,1\}$



(d) Game $\mathcal{R}^{rmprio,\mathsf{xtr}}_{hs,\ell} \to \texttt{G-M-Pr-io}^{(esalt,\ell),\mathsf{xtr},b,R,D}_{\{(hs,\ell)\}}$ for $b \in \{0,1\}$, reduction $\mathcal{R}^{rmprio,\mathsf{xtr}}_{hs,\ell}$ in grey.

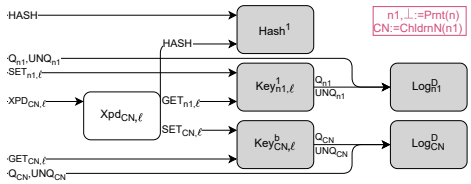


(e) Game $\texttt{Gxtr3}^b_{as,\ell}$ for $b \in \{0,1\}$

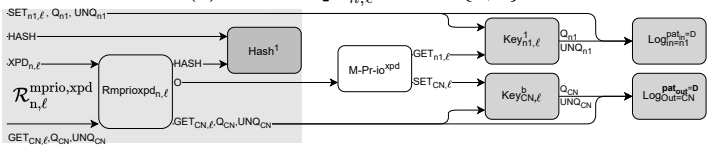


(f) Game $\mathcal{R}^{rmprio,\mathsf{xtr}}_{as,\ell} \to \texttt{G-M-Pr-io}^{(hsalt,\ell),\mathsf{xtr},b,D,D}_{\{(as,\ell)\}}$, $b \in \{0,1\}$, reduction $\mathcal{R}^{rmprio,\mathsf{xtr}}_{as,\ell}$ in grey.

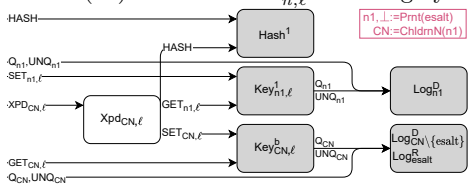
Fig. 49: Reductions for Lemma E.6



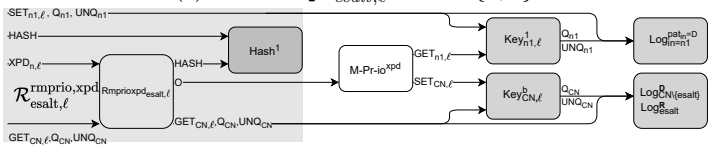
(a) Game $\texttt{Gxpd}^b_{n,\ell}$ for $b \in \{0,1\}$



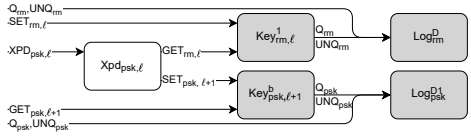
(b) Game $\mathcal{R}^{\text{mprio,xpd}}_{n,\ell} \rightarrow \texttt{G-M-Pr-io}^{(n_1,\ell),\text{xpd},b,D,D_{CN}}_{CN,\ell}$ for $b \in \{0,1\}$, where $(n_1, \_) = \mathsf{PrntN}(n)$ and $CN = \mathsf{ChldrnN}(n_1)$. Reduction $\mathcal{R}^{\text{mprio,xpd}}_{n,\ell}$ in grey.



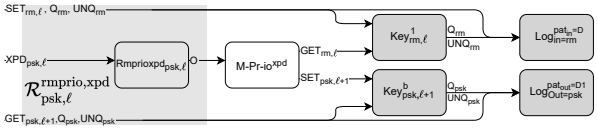
(c) Game $\texttt{Gxpd}^b_{esalt,\ell}$ for $b \in \{0,1\}$



(d) Game $\mathcal{R}^{\text{mprio,xpd}}_{esalt,\ell} \rightarrow \texttt{G-M-Pr-io}^{(n_1,\ell),\text{xpd},b,D,(D_{CN\setminus\{esalt\}}||R_{esalt})}_{CN,\ell}$ for $b \in \{0,1\}$, where $(n_1, \_) = \mathsf{PrntN}(esalt)$ and $CN = \mathsf{ChldrnN}(n_1)$. Reduction $\mathcal{R}^{\text{mprio,xpd}}_{esalt,\ell}$ in grey.



(e) Game $\texttt{Gxpd}^b_{psk,\ell}$ for $b \in \{0,1\}$



(f) Game $\mathcal{R}^{\text{mprio,xpd}}_{psk,\ell} \rightarrow \texttt{G-M-Pr-io}^{(rm,\ell),\text{xpd},b,D,D1}_{\{(psk,\ell+1)\}}$ for $b \in \{0,1\}$. Reduction $\mathcal{R}^{\text{mprio,xpd}}_{psk,\ell}$ in grey.

Fig. 50: Reductions for Lemma E.6

$\mathtt{Rmprioxtr}_{es,\ell}$	$\mathtt{Rmprioxtr}_{es,\ell} \to \mathtt{M\text{-}Pr\text{-}io}^{\mathsf{xtr}^\dagger}$	$\mathtt{Xtr}^0_{es,\ell}$
$\mathsf{XTR}_{es,\ell}(h_1, h_2)$	$\mathsf{XTR}_{es,\ell}(h_1, h_2)$	$\mathsf{XTR}_{es,\ell}(h_1, h_2)$
$n_1, n_2 \leftarrow \mathsf{PrntN}(es)$	$n_1, n_2 \leftarrow \mathsf{PrntN}(es)$	$n_1, n_2 \leftarrow \mathsf{PrntN}(es)$
if $alg(h_1) \neq \bot$	**if** $alg(h_1) \neq \bot$	**if** $\mathsf{alg}(h_1) \neq \bot$
$\wedge\, alg(h_2) \neq \bot$:	$\wedge\, alg(h_2) \neq \bot$:	$\wedge\, \mathsf{alg}(h_2) \neq \bot$:
assert $alg(h_1) = alg(h_2)$	**assert** $alg(h_1) = alg(h_2)$	**assert** $\mathsf{alg}(h_1) = \mathsf{alg}(h_2)$
$h \leftarrow \mathsf{xtr}\langle es, h_1, h_2\rangle$	$h \leftarrow \mathsf{xtr}\langle es, h_1, h_2\rangle$	$h \leftarrow \mathsf{xtr}\langle es, h_1, h_2\rangle$
$(k_1, hon_1) \leftarrow \mathsf{GET}_{n_1,\ell}(h_1)$	$(k_1, hon_1) \leftarrow \mathsf{GET}_{n_1,\ell}(h_1)$	$(k_1, hon_1) \leftarrow \mathsf{GET}_{n_1,\ell}(h_1)$
$() \leftarrow O(h_2, k_1)$	$(k_2, hon_2) \leftarrow \mathsf{GET}_{n_2,\ell}(h_2)$	$(k_2, hon_2) \leftarrow \mathsf{GET}_{n_2,\ell}(h_2)$
	$k \leftarrow \mathsf{xtr}^\dagger(k_2, k_1)$	$k \leftarrow \mathsf{xtr}(k_1, k_2)$
		$hon \leftarrow hon_1 \vee hon_2$
	$h_{out} \leftarrow \mathsf{xtr}\langle es, k_1, h_2\rangle$	**if** $b \wedge hon_2$:
	$i \leftarrow \mathsf{OutIndex}(h_{out})$	$k^\star \leftarrow\!\!\$\ \{0,1\}^{\mathsf{len}(k)}$
	assert $i \in \{(es, \ell)\}$	$k \leftarrow \mathsf{tag}_{\mathsf{alg}(k)}(k^\star)$
	$\mathsf{SET}_i(h_{out}, hon_2, k_{out})$	$h \leftarrow \mathsf{SET}_{es,\ell}(h, hon, k)$
return h	**return** h	**return** h
		$\mathtt{Id}_{\{\mathsf{GET}_{es,\ell}, \mathsf{UNQ}_{es}, \mathsf{Q}_{es}\}}$
$\mathsf{GET}_{es,\ell}(h)$	$\mathsf{GET}_{es,\ell}(h)$	$\mathsf{GET}_{es,\ell}(h)$
parse $\mathsf{xtr}\langle es, h_1, h_2\rangle = h$	**parse** $\mathsf{xtr}\langle es, h_1, h_2\rangle = h$	
$(k_1, \_) \leftarrow \mathsf{GET}_{0salt,\ell}(h_1)$	$(k_1, \_) \leftarrow \mathsf{GET}_{0salt,\ell}(h_1)$	
$h' \leftarrow \mathsf{xtr}\langle es, k_1, h_2\rangle$	$h' \leftarrow \mathsf{xtr}\langle es, k_1, h_2\rangle$	
return $\mathsf{GET}_{es,\ell}(h')$	**return** $\mathsf{GET}_{es,\ell}(h')$	**return** $\mathsf{GET}_{es,\ell}(h)$
$\mathsf{UNQ}_{es}(h)$	$\mathsf{UNQ}_{es}(h)$	$\mathsf{UNQ}_{es}(h)$
parse $\mathsf{xtr}\langle es, h_1, h_2\rangle = h$	**parse** $\mathsf{xtr}\langle es, h_1, h_2\rangle = h$	
$(k_1, \_) \leftarrow \mathsf{GET}_{0salt,\ell}(h_1)$	$(k_1, \_) \leftarrow \mathsf{GET}_{0salt,\ell}(h_1)$	
$h' \leftarrow \mathsf{xtr}\langle es, k_1, h_2\rangle$	$h' \leftarrow \mathsf{xtr}\langle es, k_1, h_2\rangle$	
return $\mathsf{UNQ}_{es}(h')$	**return** $\mathsf{UNQ}_{es}(h')$	**return** $\mathsf{UNQ}_{es}(h)$
$\mathsf{Q}_{es}(h)$	$\mathsf{Q}_{es}(h)$	$\mathsf{Q}_{es}(h)$
parse $\mathsf{xtr}\langle es, h_1, h_2\rangle = h$	**parse** $\mathsf{xtr}\langle es, h_1, h_2\rangle = h$	
$(k_1, \_) \leftarrow \mathsf{GET}_{0salt,\ell}(h_1)$	$(k_1, \_) \leftarrow \mathsf{GET}_{0salt,\ell}(h_1)$	
$h' \leftarrow \mathsf{xtr}\langle es, k_1, h_2\rangle$	$h' \leftarrow \mathsf{xtr}\langle es, k_1, h_2\rangle$	
return $\mathsf{Q}_{es}(h')$	**return** $\mathsf{Q}_{es}(h')$	**return** $\mathsf{Q}_{es}(h)$

Fig. 51: $\mathtt{Gxtr1}^b_{es,\ell} \overset{\text{code}}{\equiv} \mathcal{R}^{rmprio,\mathsf{xtr}}_{es,\ell} \to \mathtt{G\text{-}M\text{-}Pr\text{-}io}^{(psk,\ell),\mathsf{xtr}^\dagger,b,D1,D}_{\{es\}}$ code equivalence.

$\texttt{Rmprioxtr}_{hs,\ell}$

$\mathsf{XTR}_{hs,\ell}(h_1, h_2)$

$n_1, n_2 \leftarrow PrntN(hs)$
if $alg(h_1) \neq \bot$
  $\wedge\, alg(h_2) \neq \bot$:
  assert $alg(h_1) = alg(h_2)$
$h \leftarrow \mathsf{xtr}\langle hs, h_1, h_2\rangle$
$(k_2, hon_2) \leftarrow \mathsf{GET}_{n_2,\ell}(h_2)$
if hon_2 :
  $hon \leftarrow 1$
  $k \leftarrow\!\!\$\ \{0,1\}^{\mathsf{len}(h)}$
  $k^* \leftarrow \mathsf{tag}_h(k)$
  assert $\mathsf{name}(h) = hs$
  assert $\mathsf{level}(h) = \ell$
  assert $\mathsf{alg}(k^\star) = \mathsf{alg}(h)$
  $k \leftarrow \mathsf{untag}(k^\star)$
  assert $\mathsf{len}(h) = |k|$
  if $\mathsf{Q}_{hs}(h) \neq \bot$:
    return $\mathsf{Q}_n(h)$
  if b :
    if hon :
      $k \leftarrow\!\!\$\ \{0,1\}^{\mathsf{len}(h)}$
  $h' \leftarrow \mathsf{UNQ}_n(h, hon, k)$
  if $h' \neq h$:
    return h'
  $K_{n,\ell}[h] \leftarrow (k, hon)$
else
$() \leftarrow O(h_1, k_2)$
return h

$\mathsf{GET}_{hs,\ell}(h)$

parse $\mathsf{xtr}\langle hs, h_1, h_2\rangle = h$
$(k_2, hon_2) \leftarrow \mathsf{GET}_{n_2,\ell}(h_2)$
if $hon2$:
  assert $K_{n,\ell}[h] \neq \bot$
  $(hon, k) \leftarrow K_{n,\ell}[h]$
else
  $h' \leftarrow \mathsf{xtr}\langle hs, h_1, k_2\rangle$
  $(k, hon) \leftarrow \mathsf{GET}(h')$
return (k, hon)

$\mathsf{UNQ}_{hs}(h, hon, k)$

parse $\mathsf{xtr}\langle hs, h_1, h_2\rangle = h$
$(k_2, hon_2) \leftarrow \mathsf{GET}_{n_2,\ell}(h_2)$
$h' \leftarrow \mathsf{xtr}\langle hs, h_1, k_2\rangle$
if $hon2$: $\mathsf{UNQ}_{hs,\ell}(h, hon, k)$
else : $\mathsf{UNQ}_{hs,\ell}(h', hon, k)$
return h

$\texttt{Rmprioxtr}_{hs,\ell} \rightarrow \texttt{M-Pr-io}^{\mathsf{xtr}}$

$\mathsf{XTR}_{hs,\ell}(h_1, h_2)$

$n_1, n_2 \leftarrow PrntN(hs)$
if $alg(h_1) \neq \bot$
  $\wedge\, alg(h_2) \neq \bot$:
  assert $alg(h_1) = alg(h_2)$
$h \leftarrow \mathsf{xtr}\langle hs, h_1, h_2\rangle$
$(k_2, hon_2) \leftarrow \mathsf{GET}_{n_2,\ell}(h_2)$
if hon_2 :
  $hon \leftarrow 1$
  $k \leftarrow\!\!\$\ \{0,1\}^{\mathsf{len}(h)}$
  $k^* \leftarrow \mathsf{tag}_h(k)$
  assert $\mathsf{name}(h) = hs$
  assert $\mathsf{level}(h) = \ell$
  assert $\mathsf{alg}(k^\star) = \mathsf{alg}(h)$
  $k \leftarrow \mathsf{untag}(k^\star)$
  assert $\mathsf{len}(h) = |k|$
  if $\mathsf{Q}_{hs}(h) \neq \bot$:
    return $\mathsf{Q}_n(h)$
  if b :
    if hon :
      $k \leftarrow\!\!\$\ \{0,1\}^{\mathsf{len}(h)}$
  $h' \leftarrow \mathsf{UNQ}_n(h, hon, k)$
  if $h' \neq h$:
    return h'
  $K_{n,\ell}[h] \leftarrow (k, hon)$
else
  $(k_1, hon_1) \leftarrow \mathsf{GET}_{n_2,\ell}(h_1)$
  $k_{out} \leftarrow \mathsf{xtr}(k_1, k_2)$
  $h_{out} \leftarrow \mathsf{xtr}\langle hs, h_1, k_2\rangle$
  $i \leftarrow \mathsf{OutIndex}(h_{out})$
  assert $i \in \{(hs, \ell)\}$
  $() \leftarrow \mathsf{SET}_i(h_{out}, hon_1, k_{out})$
return h

$\mathsf{GET}_{hs,\ell}(h)$

Unchanged from the left

$\mathsf{UNQ}_{hs}(h)$

Unchanged from the left

$\mathsf{Q}_{hs}(h)$

Unchanged from the left

$\texttt{Rmprioxtr}_{hs,\ell} \rightarrow \texttt{M-Pr-io}^{\mathsf{xtr}}$

$\mathsf{XTR}_{hs,\ell}(h_1, h_2)$

$n_1, n_2 \leftarrow PrntN(hs)$
if $alg(h_1) \neq \bot$
  $\wedge\, alg(h_2) \neq \bot$:
  assert $alg(h_1) = alg(h_2)$
$h \leftarrow \mathsf{xtr}\langle hs, h_1, h_2\rangle$
$(k_1, hon_1) \leftarrow \mathsf{GET}_{n_2,\ell}(h_1)$
$(k_2, hon_2) \leftarrow \mathsf{GET}_{n_2,\ell}(h_2)$

$hon \leftarrow hon_1 \vee hon_2$
if hon_2 :
  $k^* \leftarrow\!\!\$\ \{0,1\}^{\mathsf{len}(h)}$
  $k \leftarrow \mathsf{tag}_h(k^*)$
  $() \leftarrow \mathsf{SET}_{hs,\ell}(h, hon, k)$
else

  $k \leftarrow \mathsf{xtr}(k_1, k_2)$
  $h_{out} \leftarrow \mathsf{xtr}\langle hs, h_1, h_2\rangle$

  $() \leftarrow \mathsf{SET}_{hs,\ell}(h_{out}, hon, k)$
return h

$\mathsf{GET}_{hs,\ell}(h)$

return $\mathsf{GET}_{hs,\ell}(h)$

$\mathsf{UNQ}_{hs}(h)$

return $\mathsf{UNQ}_{hs}(h)$

$\mathsf{Q}_{hs}(h)$

return $\mathsf{Q}_{hs}(h)$

$\texttt{Xtr}^1_{hs,\ell}$

$\mathsf{XTR}_{hs,\ell}(h_1, h_2)$

$n_1, n_2 \leftarrow \mathsf{PrntN}(n)$
if $\mathsf{alg}(h_1) \neq \bot \wedge \mathsf{alg}(h_2) \neq \bot$:
  assert $\mathsf{alg}(h_1) = \mathsf{alg}(h_2)$
$h \leftarrow \mathsf{xtr}\langle n, h_1, h_2\rangle$
$(k_1, hon_1) \leftarrow \mathsf{GET}_{n_1,\ell}(h_1)$
$(k_2, hon_2) \leftarrow \mathsf{GET}_{n_2,\ell}(h_2)$
$k \leftarrow \mathsf{xtr}(k_1, k_2)$
$hon \leftarrow hon_1 \vee hon_2$
if $1 \wedge hon_2$:
  $k^\star \leftarrow\!\!\$\ \{0,1\}^{\mathsf{len}(k)}$
  $k \leftarrow \mathsf{tag}_{\mathsf{alg}(k)}(k^\star)$
$h \leftarrow \mathsf{SET}_{hs,\ell}(h, hon, k)$
return h

$\mathtt{Id}_{\{\mathsf{GET}_{hs,\ell}, \mathsf{UNQ}_{hs}, \mathsf{Q}_{hs}\}}$

$\mathsf{GET}_{hs,\ell}(h)$

return $\mathsf{GET}_{hs,\ell}(h)$

$\mathsf{UNQ}_{hs}(h)$

return $\mathsf{UNQ}_{hs}(h)$

$\mathsf{Q}_{hs}(h)$

return $\mathsf{Q}_{hs}(h)$

Fig. 52: Code Equivalence: $\texttt{Gxtr2}^1_{hs,\ell} \overset{\text{code}}{\equiv} \mathcal{R}^{rmprio,\mathsf{xtr}}_{hs,\ell} \rightarrow \texttt{G-M-Pr-io}^{(esalt,\ell),\mathsf{xtr},b,R,D}_{\{hs\}}$

$\underline{\texttt{Rmprioxtr}_{as,\ell}}$	$\underline{\texttt{Rmprioxtr}_{as,\ell} \to \texttt{M-Pr-io}^{\mathsf{xtr}}}$	$\underline{\texttt{Xtr}^0_{as,\ell}}$
$\mathsf{XTR}_{as,\ell}(h_1, h_2)$	$\mathsf{XTR}_{as,\ell}(h_1, h_2)$	$\mathsf{XTR}_{as,\ell}(h_1, h_2)$
$n_1, n_2 \leftarrow \mathsf{PrntN}(as)$	$n_1, n_2 \leftarrow \mathsf{PrntN}(as)$	$n_1, n_2 \leftarrow \mathsf{PrntN}(as)$
if $alg(h_1) \neq \bot \wedge alg(h_2) \neq \bot$:	**if** $alg(h_1) \neq \bot \wedge alg(h_2) \neq \bot$:	**if** $\mathsf{alg}(h_1) \neq \bot \wedge \mathsf{alg}(h_2) \neq \bot$:
assert $alg(h_1) = alg(h_2)$	**assert** $alg(h_1) = alg(h_2)$	**assert** $\mathsf{alg}(h_1) = \mathsf{alg}(h_2)$
$h \leftarrow \mathsf{xtr}\langle as, h_1, h_2\rangle$	$h \leftarrow \mathsf{xtr}\langle as, h_1, h_2\rangle$	$h \leftarrow \mathsf{xtr}\langle as, h_1, h_2\rangle$
$(k_2, hon_2) \leftarrow \mathsf{GET}_{n_2,\ell}(h_2)$	$(k_2, hon_2) \leftarrow \mathsf{GET}_{n_2,\ell}(h_2)$	$(k_1, hon_1) \leftarrow \mathsf{GET}_{n_1,\ell}(h_1)$
$() \leftarrow O(h_1, k_2)$	$(k_1, hon_1) \leftarrow \mathsf{GET}_{n_1,\ell}(h_1)$	$(k_2, hon_2) \leftarrow \mathsf{GET}_{n_2,\ell}(h_2)$
	$k \leftarrow \mathsf{xtr}(k_1, k_2)$	$k \leftarrow \mathsf{xtr}(k_1, k_2)$
		$hon \leftarrow hon_1 \vee hon_2$
	$h_{out} \leftarrow \mathsf{xtr}\langle as, h_1, k_2\rangle$	**if** $b \wedge hon_2$:
	$i \leftarrow \mathsf{OutIndex}(h_{out})$	$k^\star \leftarrow\!\!\$\ \{0,1\}^{\mathsf{len}(k)}$
	assert $i \in \{(as, \ell)\}$	$k \leftarrow \mathsf{tag}_{\mathsf{alg}(k)}(k^\star)$
	$\mathsf{SET}_i(h_{out}, hon_1, k_{out})$	$h \leftarrow \mathsf{SET}_{as,\ell}(h, hon, k)$
return h	**return** h	**return** h
		$\underline{\texttt{Id}_{\{\mathsf{GET}_{as,\ell}, \mathsf{UNQ}_{as}, \mathsf{Q}_{as}\}}}$
$\mathsf{GET}_{as,\ell}(h)$	$\mathsf{GET}_{as,\ell}(h)$	$\mathsf{GET}_{as,\ell}(h)$
parse $\mathsf{xtr}\langle as, h_1, h_2\rangle = h$	**parse** $\mathsf{xtr}\langle as, h_1, h_2\rangle = h$	
$(k_2, \_) \leftarrow \mathsf{GET}_{0ikm,\ell}(h_2)$	$(k_2, \_) \leftarrow \mathsf{GET}_{0ikm,\ell}(h_2)$	
$h' \leftarrow \mathsf{xtr}\langle as, h_1, k_2\rangle$	$h' \leftarrow \mathsf{xtr}\langle as, h_1, k_2\rangle$	
return $\mathsf{GET}_{as,\ell}(h')$	**return** $\mathsf{GET}_{as,\ell}(h')$	**return** $\mathsf{GET}_{as,\ell}(h)$
$\mathsf{UNQ}_{as}(h)$	$\mathsf{UNQ}_{as}(h)$	$\mathsf{UNQ}_{as}(h)$
parse $\mathsf{xtr}\langle as, h_1, h_2\rangle = h$	**parse** $\mathsf{xtr}\langle as, h_1, h_2\rangle = h$	
$(k_2, \_) \leftarrow \mathsf{GET}_{0ikm,\ell}(h_2)$	$(k_2, \_) \leftarrow \mathsf{GET}_{0ikm,\ell}(h_2)$	
$h' \leftarrow \mathsf{xtr}\langle as, h_1, k_2\rangle$	$h' \leftarrow \mathsf{xtr}\langle as, h_1, k_2\rangle$	
return $\mathsf{UNQ}_{as}(h')$	**return** $\mathsf{UNQ}_{as}(h')$	**return** $\mathsf{UNQ}_{as}(h)$
$\mathsf{Q}_{as}(h)$	$\mathsf{Q}_{as}(h)$	$\mathsf{Q}_{as}(h)$
parse $\mathsf{xtr}\langle as, h_1, h_2\rangle = h$	**parse** $\mathsf{xtr}\langle as, h_1, h_2\rangle = h$	
$(k_2, \_) \leftarrow \mathsf{GET}_{0ikm,\ell}(h_2)$	$(k_2, \_) \leftarrow \mathsf{GET}_{0ikm,\ell}(h_2)$	
$h' \leftarrow \mathsf{xtr}\langle as, h_1, k_2\rangle$	$h' \leftarrow \mathsf{xtr}\langle as, h_1, k_2\rangle$	
return $\mathsf{Q}_{as}(h')$	**return** $\mathsf{Q}_{as}(h')$	**return** $\mathsf{Q}_{as}(h)$

Fig. 53: Code Equivalence: $\texttt{Gxtr3}^b_{es,\ell} \overset{\text{code}}{\equiv} \mathcal{R}^{rmprio,\mathsf{xtr}}_{as,\ell} \to \texttt{G-M-Pr-io}^{(hsalt,\ell),\mathsf{xtr},b,D,D}_{\{as\}}$

$\texttt{Rmprioxpd}_{n,\ell}$	$\texttt{Rmprioxpd}_{n,\ell} \to \texttt{M-Pr-io}^{\mathsf{xpd}}$	$\texttt{Xpd}_{n,\ell}$
$\mathsf{XPD}_{n,\ell}(h_1, r, args)$	$\mathsf{XPD}_{n,\ell}(h_1, r, args)$	$\mathsf{XPD}_{n,\ell}(h_1, r, args)$
$n_1, \_ \leftarrow \mathsf{PrntN}(n)$	$n_1, \_ \leftarrow \mathsf{PrntN}(n)$	$n_1, \_ \leftarrow \mathsf{PrntN}(n)$
$label \leftarrow \mathsf{Labels}(n, r)$	$label \leftarrow \mathsf{Labels}(n, r)$	$label \leftarrow \mathsf{Labels}(n, r)$
$h \leftarrow \mathsf{xpd}\langle n, label, h_1, args\rangle$	$h \leftarrow \mathsf{xpd}\langle n, label, h_1, args\rangle$	$h \leftarrow \mathsf{xpd}\langle n, label, h_1, args\rangle$
$d \leftarrow \mathsf{HASH}(args)$	$d \leftarrow \mathsf{HASH}(args)$	$(k_1, hon) \leftarrow \mathsf{GET}_{n_1,\ell}(h_1)$
$() \leftarrow O(h_1, (label, d))$	$(k_1, hon) \leftarrow \mathsf{GET}_{n_1,\ell}(h_1)$	**if** $n = psk$:
		$\ell \leftarrow \ell + 1$
	$k \leftarrow \mathsf{xpd}(k_1, (label, d))$	$k \leftarrow \mathsf{xpd}(k_1, (label, args))$
	$h_{out} \leftarrow \mathsf{xpd}\langle n, label, h_1, d\rangle$	**else**
	$i \leftarrow \mathsf{OutIndex}(h_{out})$	$d \leftarrow \mathsf{HASH}(args)$
	assert $i \in \{(n, \ell)\}$	$k \leftarrow \mathsf{xpd}(k_1, (label, d))$
	$\mathsf{SET}_i(h_{out}, hon, k_{out})$	$h \leftarrow \mathsf{SET}_{n,\ell}(h, hon, k)$
return h	**return** h	**return** h

		$\texttt{Id}_{\{\mathsf{GET}_{n,\ell}, \mathsf{UNQ}_n, \mathsf{Q}_n\}}$
$\mathsf{GET}_{n,\ell}(h)$	$\mathsf{GET}_{n,\ell}(h)$	$\mathsf{GET}_{n,\ell}(h)$
parse $\mathsf{xpd}\langle n, label, h_1, args\rangle = h$	**parse** $\mathsf{xpd}\langle n, label, h_1, args\rangle = h$	
$d \leftarrow \mathsf{HASH}(args)$	$d \leftarrow \mathsf{HASH}(args)$	
$h' \leftarrow \mathsf{xpd}\langle n, label, h_1, d\rangle$	$h' \leftarrow \mathsf{xpd}\langle n, label, h_1, d\rangle$	
return $\mathsf{GET}_{n,\ell}(h')$	**return** $\mathsf{GET}_{n,\ell}(h')$	**return** $\mathsf{GET}_{n,\ell}(h)$
$\mathsf{UNQ}_n(h)$	$\mathsf{UNQ}_n(h)$	$\mathsf{UNQ}_n(h)$
parse $\mathsf{xpd}\langle n, label, h_1, args\rangle = h$	**parse** $\mathsf{xpd}\langle n, label, h_1, args\rangle = h$	
$d \leftarrow \mathsf{HASH}(args)$	$d \leftarrow \mathsf{HASH}(args)$	
$h' \leftarrow \mathsf{xpd}\langle n, label, h_1, d\rangle$	$h' \leftarrow \mathsf{xpd}\langle n, label, h_1, d\rangle$	
return $\mathsf{UNQ}_n(h')$	**return** $\mathsf{UNQ}_n(h')$	**return** $\mathsf{UNQ}_n(h)$
$\mathsf{Q}_n(h)$	$\mathsf{Q}_n(h)$	$\mathsf{Q}_n(h)$
parse $\mathsf{xpd}\langle n, label, h_1, args\rangle = h$	**parse** $\mathsf{xpd}\langle n, label, h_1, args\rangle = h$	
$d \leftarrow \mathsf{HASH}(args)$	$d \leftarrow \mathsf{HASH}(args)$	
$h' \leftarrow \mathsf{xpd}\langle n, label, h_1, d\rangle$	$h' \leftarrow \mathsf{xpd}\langle n, label, h_1, d\rangle$	
return $\mathsf{Q}_n(h')$	**return** $\mathsf{Q}_n(h')$	**return** $\mathsf{Q}_n(h)$

Fig. 54: Code equivalence $\texttt{Gxpd}^{b}_{n,\ell} \overset{\text{code}}{\equiv} \mathcal{R}^{\text{mprio},\mathsf{xpd}}_{n,\ell} \to \texttt{G-M-Pr-io}^{(n_1,\ell),\mathsf{xpd},b,D,D_{CN}}_{CN}$.

$\texttt{Rmprioxpd}_{psk,\ell}$	$\texttt{Rmprioxpd}_{psk,\ell} \rightarrow \texttt{M-Pr-io}^{\mathsf{xpd}}$	$\texttt{Xpd}_{psk,\ell}$	$\texttt{Xpd}_{psk,\ell}$
$\mathsf{XPD}_{psk,\ell}(h_1, r, args)$	$\mathsf{XPD}_{psk,\ell}(h_1, r, args)$	$\mathsf{XPD}_{psk,\ell}(h_1, r, args)$	$\mathsf{XPD}_{psk,\ell}(h_1, r, args)$
$n_1, \_ \leftarrow \mathsf{PrntN}(psk)$	$n_1, \_ \leftarrow \mathsf{PrntN}(psk)$	$n_1, \_ \leftarrow \mathsf{PrntN}(n)$	$n_1, \_ \leftarrow \mathsf{PrntN}(n)$
$label \leftarrow \mathsf{Labels}(psk, r)$	$label \leftarrow \mathsf{Labels}(psk, r)$	$label \leftarrow \mathsf{Labels}(n, r)$	$label \leftarrow \mathsf{Labels}(n, r)$
$h \leftarrow \mathsf{xpd}\langle psk, label, h_1, args\rangle$	$h \leftarrow \mathsf{xpd}\langle psk, label, h_1, args\rangle$	$h \leftarrow \mathsf{xpd}\langle n, label, h_1, args\rangle$	$h \leftarrow \mathsf{xpd}\langle n, label, h_1, args\rangle$
$() \leftarrow O(h_1, (label, args))$	$(k_1, hon) \leftarrow \mathsf{GET}_{n_1,\ell}(h_1)$	$(k_1, hon) \leftarrow \mathsf{GET}_{n_1,\ell}(h_1)$	$(k_1, hon) \leftarrow \mathsf{GET}_{n_1,\ell}(h_1)$
	$k \leftarrow \mathsf{xpd}(k_1, (label, args))$	$k \leftarrow \mathsf{xpd}(k_1, (label, args))$	**if** $n = psk$:
		$\ell \leftarrow \ell + 1$	$\ell \leftarrow \ell + 1$
			$k \leftarrow \mathsf{xpd}(k_1, (label, args))$
	$h_{out} \leftarrow \mathsf{xpd}\langle psk, label, h_1, args\rangle$		**else**
	$i \leftarrow \mathsf{OutIndex}(h_{out})$		$d \leftarrow \mathsf{HASH}(args)$
	assert $i \in \{(psk, \ell + 1)\}$		$k \leftarrow \mathsf{xpd}(k_1, (label, d))$
	$\mathsf{SET}_i(h_{out}, hon, k_{out})$	$h \leftarrow \mathsf{SET}_{n,\ell}(h, hon, k)$	$h \leftarrow \mathsf{SET}_{n,\ell}(h, hon, k)$
return h	**return** h	**return** h	**return** h

Fig. 55: Code Equivalence: $\texttt{Gxpd}^{b}_{psk,\ell} \overset{\text{code}}{\equiv} \mathcal{R}^{\text{mprio},\mathsf{xpd}}_{psk,\ell} \rightarrow \texttt{G-M-Pr-io}^{(rm,\ell),\mathsf{xpd},b,D,D1}_{\{(psk,\ell+1)\}}$