1 Introduction

The Transport Layer Security (TLS) protocol [?] is responsible for securing billions of Internet connections every day. Usage statistics for Google Chromefirstofone[1] and Mozilla Firefoxfirstofone[2] report that 76–98% of all web page accesses are encrypted.At the heart of TLS is an authenticated key exchange (AKE) protocol, the so-called handshake protocol, responsible for providing the parties (client and server) with a shared, symmetric key that is fresh, private and authenticated. The ensuing record layer secures data using this key. The AKE protocol of TLS is based on the SIGMA ("SIGn-and-MAc") design of Krawczyk [?] for the Internet Key Exchange (IKE) protocol [?] of IPsec [?], which generically augments an unauthenticated, ephemeral Diffie–Hellman (DH) key exchange with authenticating signatures and MACs.

Naturally, the SIGMA AKE protocol and its incarnation in TLS have been the recipients of proofs of security. We contend that these largely justify the AKE protocols in principle, but not in practice, meaning not for the parameters in actual use and at the desired or expected level of security. Our work takes steps towards filling this gap.

1.1 Qualitative and Quantitative Bounds

Let us expand on this. The protocols KE we consider are built from a cyclic group $\mathbb{G}$ in which some DH problem P is assumed to be hard, a pseudorandom function PRF and unforgeable signature and MAC schemes S and M. The target for KE is session-key security with explicit authentication as originating from [?, ?, ?]. A proof of security has both a qualitative and quantitative dimension. Qualitatively, a proof of security for the AKE protocol KE says that KE meets its target definition assuming the building blocks meet theirs, where, in either case, meeting the definition means any poly-time adversary has negligible advantage in violating it.

The quantitative dimension associates to each adversary in the security game of KE a set of resources r, representing its runtime and attack surface (e.g., the number of users and executed protocol sessions the adversary has access to). It then relates the maximum advantage of any r-resource adversary in breaking KE's security to likewise advantage functions for the building blocks through an equation of the (simplified) form

$$\mathsf{Adv}_{\mathsf{KE}}(r) \leq f_{\mathbb{G}} \cdot \mathsf{Adv}_{\mathbb{G}}^{\mathsf{P}}(r_{\mathbb{G}}) + f_{\mathsf{S}} \cdot \mathsf{Adv}_{\mathsf{S}}^{\mathsf{EUF\text{-}CMA}}(r_{\mathsf{S}}) + \ldots,$$

deriving quantitative factors f_{X} and resources r_{X} for the advantage of each building block X.

Speaking asymptotically again, when f_{X} and r_{X} are polynomial functions in r, then $\mathsf{Adv}_{\mathsf{KE}}(r)$ is negligible whenever all building blocks' advantages are. Due to the complexity of key exchange models and the challenging task of combining the right components in a secure manner, key exchange analyses (including prior work on SIGMA [?] and TLS 1.3 firstoftwo[?, ?, ?, ?, ?][?, ?, ?, ?]) indeed often remain abstract and consider only qualitative, asymptotic security bounds.

Standardized protocols like TLS in contrast have to define concrete choices for each cryptographic building block. This involves considering reasonable estimates for adversarial resources (like runtime t and number of key-exchange model queries q) and specific instances and parameters for the underlying components X. One would hope that key exchange proofs can provide guidance in making sound choices that result in the desired overall security level. Unfortunately, AKE security bounds regularly are highly non-tight, meaning that f_{X} and/or r_{X} for some components X are so large that reasonable stand-alone parameters for X yield vacuous key exchange advantages for

[1] https://transparencyreport.google.com/https/

[2] https://telemetry.mozilla.org/

Adv. resources					SIGMA		TLS 1.3	
t	$\#U$	$\#S$	Curve	Target	CK [?]	Us (Thm. 6.1)	DFGS [?]	Us (Thm. 8.1)
2^{60}	2^{20}	2^{35}	`secp256r1`	2^{-68}	$\approx 2^{-61}$	$\approx 2^{-116}$	$\approx 2^{-64}$	$\approx 2^{-116}$
2^{60}	2^{30}	2^{55}	`secp256r1`	2^{-68}	$\approx 2^{-21}$	$\approx 2^{-106}$	$\approx 2^{-24}$	$\approx 2^{-106}$
2^{60}	2^{20}	2^{35}	`x25519`	2^{-68}	$\approx 2^{-57}$	$\approx 2^{-112}$	$\approx 2^{-60}$	$\approx 2^{-112}$
2^{60}	2^{30}	2^{55}	`x25519`	2^{-68}	$\approx 2^{-17}$	$\approx 2^{-102}$	$\approx 2^{-20}$	$\approx 2^{-102}$
2^{80}	2^{20}	2^{35}	`secp256r1`	2^{-48}	$\approx 2^{-21}$	$\approx 2^{-76}$	$\approx 2^{-24}$	$\approx 2^{-76}$
2^{80}	2^{30}	2^{55}	`secp256r1`	2^{-48}	1	$\approx 2^{-66}$	1	$\approx 2^{-66}$
2^{80}	2^{20}	2^{35}	`x25519`	2^{-48}	$\approx 2^{-17}$	$\approx 2^{-72}$	$\approx 2^{-20}$	$\approx 2^{-72}$
2^{80}	2^{30}	2^{55}	`x25519`	2^{-48}	1	$\approx 2^{-62}$	1	$\approx 2^{-62}$
2^{80}	2^{20}	2^{35}	`secp384r1`	2^{-112}	$\approx 2^{-149}$	$\approx 2^{-204}$	$\approx 2^{-152}$	$\approx 2^{-204}$
2^{80}	2^{30}	2^{55}	`secp384r1`	2^{-112}	$\approx 2^{-109}$	$\approx 2^{-194}$	$\approx 2^{-112}$	$\approx 2^{-194}$

Table 1: Exemplary concrete advantages of a key exchange adversary with given resources t (running time), $\#U$ (number of users), $\#S$ (number of sessions), in breaking the security of the SIGMA and TLS 1.3 protocols when instantiated with curve `secp256r1`, `secp384r1`, or `x25519`, based on the prior bounds by Canetti-Krawczyk [?] resp. Dowling et al. [?], and the bounds we establish (Theorem 6.1 and 8.1). Target indicates the maximal advantage $t/2^b$ tolerable when aiming for the respective curve's security level ($b = 128$ resp. 192 bits); entries in red-shaded cells miss that target. See Section 9 for full details and curves `secp521r1` and `x448`.

practical parameters. While the asymptotic bound tells us that scaling up the parameters for X (say, the DDH problem [?]) will at some point result in a secure overall advantage, this causes efficiency concerns (e.g., doubling elliptic curve DH security parameters means quadrupling the cost for group operations) and hence does not happen in practice.

We illustrate in Table 1 the effects of the non-tight bounds for SIGMA and TLS 1.3 when instantiating the protocols with NIST curves `secp256r1`, `secp384r1` [?], or curve `x25519` [?] and idealizing the protocols' other components (see Section 9 for full details). Following the curves' security, we aim at a security level of 128 bits, resp. 192 bits, meaning the ratio of an adversary's runtime to its advantage should be bounded by 2^{-128}, resp. 2^{-192}. When considering the advantage of key exchange adversaries running in time t, interacting in the security game with $\#U$ users and $\#S$ sessions, we can see that previous security bounds fail to meet the targeted security level for real-world–scale parameters ($\#U$ ranging in 2^{20}–2^{30} based on 2^{27} active certificates on the Internetfirstofone[3], $\#S$ ranging in 2^{35}–2^{55} based on 2^{32} Internet users and 2^{33} daily Google searchesfirstofone[4]). In the security analysis by Canetti and Krawczyk [?] (CK) for SIGMA, the factor associated to the decisional Diffie–Hellman problem is $f_{\mathsf{DDH}}(t, \#U, \#S) = \#U \cdot \#S$, where $\#U$ and $\#S$ again are the number of users, resp. sessions, accessible by the adversary. The analysis by Dowling et al. [?] (DFGS) for TLS 1.3 reduces to the strong Diffie–Hellman problem [?]—via the PRF-ODH assumption [?, ?]—with factor $f_{\mathsf{stDH}}(t, \#U, \#S) = (\#S)^2$. In contrast, we reduce to the strong Diffie–Hellman problem with a constant factor for both SIGMA and TLS 1.3.

Let us discuss three data points from Table 1:

1. Already with medium-sized resources, investing time $t = 2^{60}$ and interacting with a million users ($\#U = 2^{20}$) and a few billion sessions ($\#S = 2^{35}$), the CK [?] and DFGS [?] advantage bounds for SIGMA and TLS 1.3 with curves `secp256r1` and `x25519` fall 6–11 bits below the target of 2^{-68} for 128-bit security.

[3] https://letsencrypt.org/stats/

[4] https://www.internetlivestats.com/

2. When considering a more powerful, global-scale adversary ($t = 2^{80}$, $\#U = 2^{30}$, $\#S = 2^{55}$), both CK and DFGS bounds for `secp256r1/x25519` become fully vacuous; the upper bound on the probability of the adversary breaking the protocol is 1. We stress that `secp256r1` is the mandatory-to-implement curve for TLS 1.3; `secp256r1` and `x25519` together make up for 90% of the TLS 1.3 ECDHE handshakes reported through Firefox Telemetry.

3. Finally, and notably, even switching to the higher-security curve `secp384r1` helps only marginally in the latter case: the resulting advantage against SIGMA falls 3 bits short of the 192-bit security target of 2^{-112}, and the TLS advantage bound only barely meets that target.

For all curves and choices of parameters, our bounds do better.

1.2 Contributions

Most prior results in tightly secure key exchange (e.g., [?, ?]) apply only to bespoke protocols, carefully designed to allow for tighter proof techniques, at the cost of requiring more complex primitives which, in the end, eat up the gained practical efficiency. Recently, Cohn-Gordon et al. [?, ?] established a proof strategy for a simple and efficient DH key exchange with reasonable tightness loss (only linear in the number of users $\#U$), achieving implicit authentication through static DH keys through careful key derivation via a random oracle [?] with an optional explicit-authentication step.

Our work in contrast establishes tight security for standardized AKE protocols. We give tight reductions for the security of SIGMA and TLS 1.3 to the strong Diffie–Hellman problem [?], which in addition we prove is as hard as the discrete logarithm problem in the generic group model (GGM) [?, ?]. Instantiating our bounds shows that, with standardized real-world parameters, we achieve the intended security levels even when considering powerful, globally-scaled attackers.

Code-based security model and proofs. For our proofs, we provide detailed proof steps and reductions using the code-based game-playing framework of Bellare and Rogaway [?]. Our security model is similar to the one applied by Cohn-Gordon et al. [?], but formalized also as a code-based game (in Section 2) and stronger in that it captures explicit authentication and regular ("perfect") forward secrecy (instead of only weak forward secrecy in [?]).

Tighter security proof of SIGMA(-I). We establish fully quantitative security bounds for SIGMA and its identity-protecting variant SIGMA-I [?] in Sections 5 and 6. Our result is for BR-like [?] key exchange security and gives a tight reduction to the strong Diffie–Hellman problem [?] in the used DH group, and to the multi-user (mu) security of the employed pseudorandom function (PRF), signature scheme, and MAC scheme, adapting the techniques by Cohn-Gordon et al. [?] in the random oracle model [?]. The latter mu-security bounds are essentially equivalent to the corresponding bounds by CK [?]. Our improvement comes from shaving off a factor of $\#U \cdot \#S$ (number of users times number of sessions) on the DH problem advantage compared to CK. While we move to the interactive strong Diffie–Hellman problem (compared to firstoftwothe decisional DH (DDH) problem [?] used in [?]DDH [?] used in [?]), we prove (in Appendix 4) that the strong DH problem, like DDH, is as hard as solving discrete logarithms in the generic group model [?, ?]firstofone, reflecting the (only generic) algorithms known for solving discrete logarithms in elliptic curve groups.

Tighter security proof for the TLS 1.3 DH handshake. We likewise establish fully quantitative security bounds for the key exchange of the recently standardized newest version of the Transport Layer Security protocol, TLS 1.3 [?], in Sections 7 and 8. The main quantitative improvement in our reduction is again a tight reduction to the strong DH problem, whereas prior bounds by DFGS [?] incurred a quadratic loss to the PRF-ODH assumption [?, ?], a loss which translates directly to strong DH [?]. While TLS 1.3 roughly follows the SIGMA-I design, its cascading key schedule impedes the precise technique of Cohn-Gordon et al. [?] and a direct application of our results on SIGMA-I, as no single function (to be modeled as a random oracle) binds the Diffie–Hellman values to the session context. We therefore have to carefully adapt the proof to accommodate the more complex key schedule and other core variations in TLS 1.3's key exchange, achieving conceptually similar tightness results as for SIGMA-I.

Evaluation. In Section 9, we evaluate the concrete security implications of our improved bounds for SIGMA and TLS 1.3 for a wide range of real-world resource parameters and all five elliptic curves firstofone(`secp256r1`, `secp384r1`, `secp521r1`,`x25519`, `x448`) standardized for use in TLS 1.3 [?], a summary of which is displayed in Table 1. Leveraging our GGM bound for the strong Diffie–Hellman problem, we focus on the hardness of solving discrete logarithms in the respective elliptic curve groups, instantiating signatures based on ECDSA [?] resp. EdDSA [?]. We idealized the symmetric PRF, MAC, and hash function primitives (in two variants, with key and output sizes twice as large as the curve's security level, or fixed at 256 bits corresponding to the choice in most TLS 1.3 cipher suites).

We report that our tighter proofs indeed materialize for a wide range of real-world resource parametersfirstofone (adversary runtime $t \in \{2^{40}, 2^{60}, 2^{80}\}$, number of users $\#U \in \{2^{20}, 2^{30}\}$, and number of sessions $\#S \in \{2^{35}, 2^{45}, 2^{55}\}$). The resulting attacker advantages meet the targeted security levels of all five curves. In comparison to the prior CK [?] SIGMA and DFGS [?] TLS 1.3 bounds, our results improve the obtained security across these real-world parameters by up to 85 bits for SIGMA and 92 bits for TLS 1.3, respectively.

1.3 Optimizations, Limitations, and Possible Extensions

SIGMA being a generic AKE design, the signature, PRF, and MAC schemes may be instantiated with primitives optimized for multi-user security. While we focus on standardized and deployed schemes in our evaluation without assuming tight mu-security, our SIGMA bound allows to directly leverage such optimization. For PRFs and MACs, efficient candidates exist (e.g., AMAC [?]). For signatures, tight mu-security is more challenging [?] and often involves computationally much more expensive constructions [?].

Like Cohn-Gordon et al. [?], our key exchange security model considers exposure of long-term secrets and session keys, but does not allow revealing internal session state or randomness (as in the (e)CK model [?, ?]). This is appropriate for protocols like TLS 1.3 not aiming to protect against such threats. The original SIGMA proof [?] did establish security in the CK model [?] allowing exposure of session state; in that sense our results are qualitatively weaker. In recent work, Jager et al. [?] give a tightly secure protocol which uses symmetric state encryption to protect against ephemeral state reveals. Establishing a tight security reduction for a SIGMA-style DH-based AKE protocol which can handle adaptive compromises of session state (including DH exponents) remains a challenging open problem.

In our proofs, we crucially rely on the ability to observe and program a random oracle used for key derivation in the AKE protocol, borrowing from [?]. Notably, the approach of Cohn-Gordon et al. is tailored to an AKE protocol achieving authenticity implicitly through mixing long-term

DH keys into the key derivation. Our proofs can hence be seen as translating and adapting their technique to the setting of SIGMA and TLS 1.3, where an unauthenticated ephemeral DH exchange is explicitly authenticated through signatures and MACs, confirming that the generic SIGMA design as well as the standardized TLS 1.3 protocol bind enough context to their DH shares for this proof technique to work. Leveraging the random oracle model [?] is another qualitative difference compared to the original SIGMA proof [?] in the standard model. Interestingly, this distinction vanishes in comparison to the provable security results for the TLS 1.3 handshake protocol [?, ?, ?, ?] which employ the PRF-ODH assumption [?, ?], an interactive assumption which plausibly can only be instantiated in the random oracle model (from the strong DH assumption).

1.4 Concurrent Work

In concurrent and independent work, Diemert and Jager (DJ) [?] studied the tight security of the main TLS 1.3 handshake. Their work also tightly reduces the security of TLS 1.3 to the strong Diffie–Hellman problem by extending the technique of Cohn-Gordon et al. [?], and their bounds and ours are similarly tight. When instantiated with real-world parameters, both bounds are dominated by the same terms, as we will demonstrate in Section 9. Our proof differs from theirs in two key ways: We use an incomparable security model that is weaker in some ways and stronger in others, and we approximate the TLS 1.3 key schedule with fewer random oracles. We also contextualize our results quite differently than the DJ work, with a detailed numerical analysis that is enabled by our fully parameterized, concrete bounds. Uniquely to this work, we treat the more generic SIGMA-I protocol and justify our use of the strong DH problem with new bounds in the generic group model. Diemert and Jager [?] in turn study tight composition with the TLS record protocol.

The DJ analysis is carried out in the multi-stage key exchange model [?], proving security not only of the final session key, but also of intermediate handshake encryption keys and further secrets. While our proof does show security of these intermediate keys, we do not treat them as first-class keys accessible to the adversary through dedicated queries in the security model. Unlike either the DJ or Cohn-Gordon et al. works, our model addresses explicit authentication, which we prove via HMAC's unforgeability.

To tackle the challenge that TLS 1.3's key schedule does not bind DH values and session context in one function, DJ model the full cascading derivation of each intermediate key monolithically as an independent, programmable random oracle (cf. [?, Theorem 6]). We instead model the key schedule's inner HKDF [?] extraction and expansion functions as two individual random oracles, carefully connected via efficient look-up tables, yielding a slightly less extensive use of random oracles and compensating for the existence of shared computations in the derivation of multiple keys. This approach produces more compact bounds and allows our analysis to stay closer to the use of HKDF in TLS 1.3, where the output of one extraction call is used to derive multiple keys.

2 AKE Security Modelfirstoftwo and Multi-User Building Blocks

We provide our results in a game-based key exchange model formalized in Figure 1, at its core following the seminal work by Bellare and Rogaway [?] considering an active network adversary that controls all communication (initiating sessions and determining their next inputs through Send queries) and is able to corrupt long-term secrets (RevLongTermKey) as well as session keys (RevSessionKey). The adversary's goal is then to (a) distinguish the established shared *session key* in a "fresh" (not trivially compromised, captured through a Fresh predicate) session from a uniformly random key obtained through Test queries (breaking *key secrecy*), or (b) make a session accept without matching communication partner (breaking *explicit authentication*).

Following Cohn-Gordon et al. [?], we formalize our model in a real-or-random version (following Abdalla, Fouque, and Pointcheval [?] with added forward secrecy [?]) with *many* TEST queries which all answer with a real or uniformly random session key based on the *same* random bit b. We focus on the security of the *main* session key established. While our proofs (for both SIGMA and TLS 1.3) establish security of the intermediate encryption and MAC keys, too, we do not treat them as first-class keys available to the adversary through TEST and REVSESSIONKEY queries. We expect that our results extend to a multi-stage key exchange (MSKE [?]) treatment and refer to the concurrent work by Diemert and Jager [?] for tight results for TLS 1.3 in a MSKE model.

In contrast to the work by Cohn-Gordon et al. [?] and Diemert and Jager [?], our model additionally captures explicit authentication through the $\mathsf{ExplicitAuth}$ predicate in Figure 1, ensuring sessions with non-corrupted peer accept with an honest partner session. We and [?] further treat protocols where the communication partner's identity of a session may be unknown at the outset and only learned during the protocol execution; this setting of "post-specified peers" [?] particularly applies to the SIGMA protocol family [?] as well as TLS 1.3 [?].

2.1 Key Exchange Protocols

We begin by formalizing the syntax of key exchange protocols. A key exchange protocol KE consists of three algorithms ($\mathsf{KGen}, \mathsf{Activate}, \mathsf{Run}$) and an associated key space $\mathsf{KE.KS}$ (where most commonly $\mathsf{KE.KS} = \{0,1\}^n$ for some $n \in \mathbb{N}$). The key generation algorithm $\mathsf{KGen}() \xrightarrow{\$} (pk, sk)$ generates new long-term public/secret key pairs. In the security model, we will associate key pairs to distinct *users* (or *parties*) with some identity $u \in \mathbb{N}$ running the protocol, and log the public long-term keys associated with each user identity in a list *peerpk*. (The adversary will be in control of initializing new users, identified by an increasing counter, and we assume it only references existing user identities.) The activation algorithm $\mathsf{Activate}(id, sk, peerid, peerpk, role) \xrightarrow{\$} (st', m')$ initiates a new session for a given user identity id (and associated long-term secret key sk) acting in a given role $role \in \{\mathsf{initiator}, \mathsf{responder}\}$ and aiming to communicate with some peer user identity *peerid*. $\mathsf{Activate}$ also takes as input the list *peerpk* of all users' public keys; protocols may use this list to look up their own and their peers' public keys. We provide the entire list instead of just the user's and peers' public keys to accommodate protocols with post-specified peer. These protocols may leave *peerid* unspecified at the time of session activation; when the peer identity is set at some later point, the list can be used to find the corresponding long-term key. Activation outputs a session state and (if $role = \mathsf{initiator}$) first protocol message m', and will be invoked in the security model to create a new session π_u^i at a user u (where the label i distinguishes different sessions of the same user). Finally, $\mathsf{Run}(id, sk, st, peerpk, m) \xrightarrow{\$} (st', m')$ delivers the next incoming key exchange message m to the session of user id with secret key sk and state st, resulting in an updated state st' and a response message m'. Like $\mathsf{Activate}$, it relies on the list *peerpk* to look up its own and its peer's long-term keys.

The state of each session in a key exchange protocol contains at least the following variables, beyond possibly further, protocol-specific information:

$peerid \in \mathbb{N}$. Reflects the (intended) partner identity of the session; firstoftwoin protocols with post-specified peers this is learned and set (once) by the session during the protocol execution. if post-specified, this is learned and set (once) during protocol execution.

$role \in \{\mathsf{initiator}, \mathsf{responder}\}$. The session's role, determined upon activation.

$status \in \{\mathsf{running}, \mathsf{accepted}, \mathsf{rejected}\}$. The session's status; initially $status = \mathsf{running}$, a session accepts when it switches to $status = \mathsf{accepted}$ (once).

skey $\in$ KE.KS. The derived session key (infirstofone the protocol-specific key space KE.KS), set upon acceptance.

sid. The session identifier used to define partnered session in the security model; initially unset, *sid* is determined (once) during protocol execution.

2.2 Key Exchange Security

We formalize our key exchange security game $G^{\mathsf{KE\text{-}SEC}}_{\mathsf{KE},\mathcal{A}}$ in Figure 1, based on the concepts introduced above in Figure 1 and following the framework for code-based game playing by Bellare and Rogaway [?]. After initializing the game, the adversary $\mathcal{A}$ is given access to queries NewUser (generating a new user's public/secret key pair), Send (controlling activation and message processing of sessions), RevSessionKey (revealing session keys), RevLongTermKey (corrupting user's long-term secrets), and Test (providing challenge real-or-random session keys), as well as a Finalize query to which it will submit its guess b' for the challenge bit b, ending the game.

The game $G^{\mathsf{KE\text{-}SEC}}_{\mathsf{KE},\mathcal{A}}$ then (in Finalize) determines whether $\mathcal{A}$ was successful through the following three predicates, formalized in pseudocode in Figure 1:

Sound. The soundness predicate Sound checks that (a) no three session identifiers collide (hence the session identifier properly serves to identify two partnered sessions). Furthermore, it ensures that (b) accepted sessions with the same session identifier, agreeing partner identities, and distinct roles derive the same session key. The adversary breaks soundness if it violates either of these properties.

ExplicitAuth. The predicate ExplicitAuth captures explicit authentication in that it requires that for any session of some user *id* that accepted while its partner *peerid* was not corrupted (captured through logging relative acceptance time $\mathsf{t_{acc}}$ and long-term reveal time revltk_{peerid}) has (a) a partnered session run by the intended peer identity and in an opposite role, and (b) if that partnered session accepts, it will do so with peer identity *id*. The adversary breaks explicit authentication if this predicate evaluates to false.

Fresh. Finally, to capture key secrecy, we have to restrict the adversary to testing only so-called *fresh* sessions in order to exclude trivial attacks, which the freshness predicate Fresh ensures. A tested session is non-fresh, if (a) its session key has been revealed (in which case $\mathcal{A}$ knows the real key), (b) its partnered session (through *sid*) has been revealed or tested (in which case $\mathcal{A}$ knows the real key or may see two different random keys), or (c) its intended peer identity was compromised prior to accepting (in which case $\mathcal{A}$ may fully control the communication partner). If the adversary violates freshness, we invalidate its guess by overwriting $b' \leftarrow 0$.

We call two distinct sessions π^i_u and π^j_v *partnered* if $\pi^i_u.sid = \pi^j_v.sid$. We refer to sessions generated by Activate (i.e., controlled by the game) as *honest* sessions to reflect that their behavior is determined honestly by the game and not the adversary. The long-term key of an honest session may still be corrupted, or its session key may be revealed without affecting this notion of "honesty".

Definition 2.1 (Key exchange security). *Let* KE *be a key exchange protocol and* $G^{\mathsf{KE\text{-}SEC}}_{\mathsf{KE},\mathcal{A}}$ *be the key exchange security game defined in Figure 1. We define*

$$\mathsf{Adv}^{\mathsf{KE\text{-}SEC}}_{\mathsf{KE}}(t, q_{\mathrm{N}}, q_{\mathrm{S}}, q_{\mathrm{RS}}, q_{\mathrm{RL}}, q_{\mathrm{T}}) := 2 \cdot \max_{\mathcal{A}} \Pr\left[\mathrm{G}^{\mathsf{KE\text{-}SEC}}_{\mathsf{KE},\mathcal{A}} \Rightarrow 1\right] - 1,$$

where the maximum is taken over all adversaries, denoted $(t, q_{\mathrm{N}}, q_{\mathrm{S}}, q_{\mathrm{RS}}, q_{\mathrm{RL}}, q_{\mathrm{T}})$-KE-SEC-*adversaries, running in time at most t and making at most q_{N}, q_{S}, q_{RS}, q_{RL}, resp. q_{T} queries to their oracles* NewUser, Send, RevSessionKey, RevLongTermKey, *resp.* Test.

$G^{\mathsf{KE\text{-}SEC}}_{\mathsf{KE},\mathcal{A}}$

```
INITIALIZE:
$\mathsf{time} \leftarrow 0$; $\mathsf{users} \leftarrow 0$
$b \xleftarrow{\$} \{0,1\}$

NEWUSER:
$\mathsf{users} \leftarrow \mathsf{users} + 1$
$(pk_{\mathsf{users}}, sk_{\mathsf{users}}) \xleftarrow{\$} \mathsf{KGen}()$
$\mathsf{revltk}_{\mathsf{users}} \leftarrow \infty$
$peerpk[\mathsf{users}] \leftarrow pk_{\mathsf{users}}$
return $pk_{\mathsf{users}}$

SEND$(u, i, m)$:
if $\pi_u^i = \bot$ then
    $(peerid, role) \leftarrow m$
    $(\pi_u^i, m') \xleftarrow{\$} \mathsf{Activate}(u, sk_u, peerid, peerpk, role)$
    $\pi_u^i.\mathsf{t_{acc}} \leftarrow 0$
else
    $(\pi_u^i, m') \xleftarrow{\$} \mathsf{Run}(u, sk_u, \pi_u^i, peerpk, m)$
if $\pi_u^i.status = \mathsf{accepted}$ then
    $\mathsf{time} \leftarrow \mathsf{time} + 1$
    $\pi_u^i.\mathsf{t_{acc}} \leftarrow \mathsf{time}$
return $m'$

REVSESSIONKEY$(u, i)$:
if $\pi_u^i = \bot$ or $\pi_u^i.status \neq \mathsf{accepted}$ then
    return $\bot$
$\pi_u^i.\mathsf{revealed} \leftarrow \mathsf{true}$
return $\pi_u^i.skey$

REVLONGTERMKEY$(u)$:
$\mathsf{time} \leftarrow \mathsf{time} + 1$
$\mathsf{revltk}_u \leftarrow \mathsf{time}$
return $sk_u$

TEST$(u, i)$:
if $\pi_u^i = \bot$ or $\pi_u^i.status \neq \mathsf{accepted}$ or $\pi_u^i.\mathsf{tested}$ then
    return $\bot$
$\pi_u^i.\mathsf{tested} \leftarrow \mathsf{true}$
$T \leftarrow T \cup \{\pi_u^i\}$
$k_0 \leftarrow \pi_u^i.skey$
$k_1 \xleftarrow{\$} \mathsf{KE.KS}$
return $k_b$
```

```
FINALIZE$(b')$:
if $\neg\mathsf{Sound}$ then
    return 1
if $\neg\mathsf{ExplicitAuth}$ then
    return 1
if $\neg\mathsf{Fresh}$ then
    $b' \leftarrow 0$
return $[[b = b']]$

Sound:
if $\exists$ distinct $\pi_u^i$, $\pi_v^j$, $\pi_w^k$ with $\pi_u^i.sid = \pi_v^j.sid = \pi_w^k.sid$
then // no triple sid match
    return $\mathsf{false}$
if $\exists \pi_u^i, \pi_v^j$ with
    $\pi_u^i.status = \pi_v^j.status = \mathsf{accepted}$
    and $\pi_u^i.sid = \pi_v^j.sid$
    and $\pi_u^i.peerid = v$ and $\pi_v^j.peerid = u$
    and $\pi_u^i.role \neq \pi_v^j.role$, but $\pi_u^i.skey \neq \pi_v^j.skey$ then
// partnering implies same key
    return $\mathsf{false}$
return $\mathsf{true}$

ExplicitAuth:
return
    $\forall \pi_u^i : \pi_u^i.status = \mathsf{accepted}$
          and $\pi_u^i.\mathsf{t_{acc}} < \mathsf{revltk}_{\pi_u^i.peerid}$
// all sessions accepting with a non-corrupted peer ...
        $\implies \exists \pi_v^j : \pi_u^i.peerid = v$
              and $\pi_u^i.sid = \pi_v^j.sid$
              and $\pi_u^i.role \neq \pi_v^j.role$
// ... have a partnered session ...
            and $(\pi_v^j.status = \mathsf{accepted} \implies \pi_v^j.peerid = u)$
// ... agreeing on the peerid (upon acceptance)

Fresh:
for each $\pi_u^i \in T$
    if $\pi_u^i.\mathsf{revealed}$ then
        return $\mathsf{false}$ // tested session may not be revealed
    if $\exists \pi_v^j \neq \pi_u^i : \pi_v^j.sid = \pi_u^i.sid$
        and $(\pi_v^j.\mathsf{tested}$ or $\pi_v^j.\mathsf{revealed})$ then
        return $\mathsf{false}$ // tested session's partnered session may not be tested or revealed
    if $\mathsf{revltk}_{\pi_u^i.peerid} < \pi_u^i.\mathsf{t_{acc}}$ then
        return $\mathsf{false}$ // tested session's peer may not be corrupted prior to acceptance
return $\mathsf{true}$
```

Figure 1: Key exchange security game.

2.3 Security Properties

Let us briefly revisit some core security properties captured in our key exchange security model.

First, we capture regular *key secrecy* of the main session key through Test queries, incorporating *forward secrecy* (sometimes called "perfect" forward secrecy) by allowing the adversary to corrupt any user as long as all tested sessions accept prior to corrupting their respective intended peer. This strengthens our model compared to that of Cohn-Gordon et al. [?] which only captures weak forward secrecy where the adversary has to be passive in sessions where it corrupts long-term secrets. Diemert and Jager [?] additionally treat the security of intermediate keys and further secrets beyond the main session key in a multi-stage approach [?], but without capturing explicit authentication.

Our model encodes *explicit authentication* (via ExplicitAuth), a strengthening compared to the implicit-authentication model in [?].

Like [?, ?], our model captures *key-compromise impersonation* attacks by allowing the session owner's secret key of tested sessions to be corrupted at any point in time. Similarly, we do *not* capture *session-state or randomness reveals* [?, ?] or *post-compromise security* [?].

3 Assumptions, Building Blocks, firstofoneand Multi-User Security

Before we continue to our main technical results, let us briefly introduce notation and discuss the multi-user security of the involved building blocks: strong Diffie–Hellman (including the GGM bound we prove), PRFs, digital signatures, MAC schemes, and hash functions.

3.1 Decisional and Strong Diffie–Hellman

The classical decisional Diffie–Hellman assumption [?] states that, when only observing the two Diffie–Hellman shares g^x, g^y, the resulting secret g^{xy} is indistinguishable from a random group element.

Definition 3.1 (Decisional Diffie–Hellman (DDH) assumption)**.** *Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order p. We define*

$$\mathsf{Adv}^{\mathsf{DDH}}_{\mathbb{G}}(t) := \max_{\mathcal{A}} \Big| \Pr\left[\mathcal{A}(\mathbb{G}, g, g^x, g^y, g^{xy}) \Rightarrow 1 \mid x, y \xleftarrow{\$} \mathbb{Z}_p\right] - \Pr\left[\mathcal{A}(\mathbb{G}, g, g^x, g^y, g^z) \Rightarrow 1 \mid x, y, z \xleftarrow{\$} \mathbb{Z}_p\right] \Big|,$$

where the maximum is taken over all adversaries, denoted (t)-DDH-adversaries *running in time at most t.*

The strong Diffie–Hellman assumption, a weakening of the gap Diffie–Hellman assumption [?], states that solving the computational Diffie–Hellman problem given a restricted decisional Diffie–Hellman oracle is hard.

Definition 3.2 (Strong Diffie–Hellman assumption [?])**.** *Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order p. Let $\mathsf{DDH}(X, Y, Z) := [[X^{\log_g(Y)} = Z]]$ be a decisional Diffie–Hellman oracle. We define*

$$\mathsf{Adv}^{\mathsf{stDH}}_{\mathbb{G}}(t, q_{\mathrm{SDH}}) := \max_{\mathcal{A}} \Pr\left[\mathcal{A}^{\mathsf{DDH}(g^x,\cdot,\cdot)}(\mathbb{G}, g, g^x, g^y) = g^{xy} \;\Big|\; x, y \xleftarrow{\$} \mathbb{Z}_p\right],$$

where the maximum is taken over all adversaries, denoted (t, q_{SDH})-stDH-adversaries *running in time at most t and making at most q_{SDH} queries to their* DDH *oracle.*

The strong (or gap) Diffie–Hellman assumption has been deployed in numerous works to analyze practical key exchange designs, directly or through the PRF-ODH assumption [?, ?] it supports, including [?, ?, ?, ?, ?, ?, ?] as well as in the closely related works on practical tightness by Cohn-Gordon et al. [?] and Diemert and Jager [?]. To argue that it is reasonable to rely on the strong Diffie–Hellman assumption, we turn to the generic group model [?, ?]. Although some known algorithms for solving discrete logarithms in finite fields like index calculus fall outside the generic group model, the best known algorithms for elliptic curve groups are generic. Shoup [?] proved that, in the generic group model, any adversary computing at most t group operations in a group of prime order p has advantage at most $\mathcal{O}(t^2/p)$ in solving the discrete logarithm problem or the computational or decisional Diffie–Hellman problem in that group. We claim, and provefirstofone in Appendix 4, that any adversary in the generic group model making at most t group operations and DDH oracle queries, also has advantage at most $\mathcal{O}(t^2/p)$ in solving the strong Diffie–Hellman problem.

Theorem 3.3. *Let $\mathbb{G}$ be a group with prime order p. In the generic group model, $\mathsf{Adv}_{\mathbb{G}}^{\mathsf{stDH}}(t,q) \leq 4t^2/p$.*

3.2 Multi-User PRF Security

Let us recap the multi-user security notion for pseudorandom functions (PRFs) [?].

Definition 3.4 (Multi-user PRF security)**.** *Let $\mathsf{PRF}\colon \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^n$ be a function (for $k,n \in \mathbb{N}$ and $m \in \mathbb{N} \cup \{*\}$) and $\mathrm{G}_{\mathsf{PRF},\mathcal{A}}^{\mathsf{mu\text{-}PRF}}$ be the multi-user PRF security game defined as in Figure 2. We define*

$$\mathsf{Adv}_{\mathsf{PRF}}^{\mathsf{mu\text{-}PRF}}(t, q_{\mathrm{Nw}}, q_{\mathrm{Fn}}, q_{\mathrm{Fn/U}}) := 2 \cdot \max_{\mathcal{A}} \Pr\left[\mathrm{G}_{\mathsf{PRF},\mathcal{A}}^{\mathsf{mu\text{-}PRF}} \Rightarrow 1\right] - 1,$$

where the maximum is taken over all adversaries, denoted $(t, q_{\mathrm{Nw}}, q_{\mathrm{Fn}}, q_{\mathrm{Fn/U}})$-$\mathsf{mu\text{-}PRF}$-adversaries, running in time at most t and making at most q_{Nw} queries to their NEW *oracle, at most q_{Fn} total queries to their* FN *oracle, and at most $q_{\mathrm{Fn/U}}$ queries* $\mathrm{F}\text{N}(i,\cdot)$ *for any user i.*

Generically, the multi-user security of PRFs reduces to single-user security (formally, $\mathrm{G}_{\mathsf{PRF},\mathcal{A}}^{\mathsf{mu\text{-}PRF}}$ with $\mathcal{A}$ restricted to $q_{\mathrm{Nw}} = 1$ queries to NEW) with a factor in the number of users via a hybrid argument [?], i.e.,

$$\mathsf{Adv}_{\mathsf{PRF}}^{\mathsf{mu\text{-}PRF}}(t, q_{\mathrm{Nw}}, q_{\mathrm{Fn}}, q_{\mathrm{Fn/U}}) \leq q_{\mathrm{Nw}} \cdot \mathsf{Adv}_{\mathsf{PRF}}^{\mathsf{mu\text{-}PRF}}(t', 1, q_{\mathrm{Fn/U}}, q_{\mathrm{Fn/U}}),$$

where $t \approx t'$. (Note that the total number q_{Fn} of queries to the FN oracle across all users does not affect the reduction.) There exist simple and efficient constructions, like AMAC [?], that however achieve multi-user security tightly.

If we use a random oracle RO as a PRF with key length kl, then

$$\mathsf{Adv}_{\mathsf{RO}}^{\mathsf{mu\text{-}PRF}}(t, q_{\mathrm{Nw}}, q_{\mathrm{Fn}}, q_{\mathrm{Fn/U}}, q_{\mathrm{RO}}) \leq \frac{q_{\mathrm{Nw}} \cdot q_{\mathrm{RO}}}{2^{kl}}.$$

3.3 Multi-User Unforgeability with Adaptive Corruptions of Signatures and MACs

We recap the definition of digital signature schemes and message authentication codes (MACs) as well as the natural extension of classical *existential unforgeability under chosen-message attacks* [?] to the *multi-user* setting with *adaptive corruptions.* For signatures, this notion was considered by Bader et al. [?] and, without corruptions, by Menezes and Smart [?].

```
$\mathrm{G}^{\mathsf{mu\text{-}PRF}}_{\mathsf{PRF},\mathcal{A}}$

INITIALIZE:
  $b \xleftarrow{\$} \{0,1\}$
  $u \leftarrow 0$

NEW:
  $u \leftarrow u + 1$
  if $b = 1$ then
      $K_u \xleftarrow{\$} \{0,1\}^k$ firstoftwo
      ; $f_u := \mathsf{PRF}(K_u, \cdot)$
  else firstofone
      $f_u \xleftarrow{\$} \mathsf{FUNC}$

FN($i, x$):
  return $f_i(x)$

FINALIZE($b^*$):
  return $[[b = b^*]]$
```

Figure 2: Multi-user PRF security of a pseudorandom function $\mathsf{PRF}\colon \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^n$. FUNC is the space of all functions $\{0,1\}^m \to \{0,1\}^n$.

Definition 3.5 (Signature scheme)**.** *A* signature scheme $\mathsf{S} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ *consists of three efficient algorithms defined as follows.*

- $\mathsf{KGen}() \xrightarrow{\$} (pk, sk)$*. This probabilistic algorithm generates a public verification key pk and a secret signing key sk.*
- $\mathsf{Sign}(sk, m) \xrightarrow{\$} \sigma$*. On input a signing key sk and a message m, this (possibly) probabilistic algorithm outputs a signature σ.*
- $\mathsf{Vrfy}(vk, m, \sigma) \to d$*. On input a verification key pk, a message m, and a signature σ, this deterministic algorithm outputs a decision bit* $d \in \{0,1\}$ *(where* $d = 1$ *indicates validity of the signature).*

Definition 3.6 (Signature mu-EUF-CMA security)**.** *Let* S *be a signature scheme and* $\mathrm{G}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{S},\mathcal{A}}$ *be the game for signature multi-user existential unforgeability under chosen-message attacks with adaptive corruptions firstoftwodefined as in Figure 3(see the full version [?] for the formal definition). We define*

$$\mathsf{Adv}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{S}}(t, q_{\mathrm{Nw}}, q_{\mathrm{SG}}, q_{\mathrm{SG/U}}, q_{\mathrm{C}}) := \max_{\mathcal{A}} \Pr\left[\mathrm{G}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{S},\mathcal{A}} \Rightarrow 1\right],$$

where the maximum is taken over all adversaries, denoted $(t, q_{\mathrm{Nw}}, q_{\mathrm{SG}}, q_{\mathrm{SG/U}}, q_{\mathrm{C}})$-mu-EUF-CMA-adversaries*, running in time at most* t *and making at most* q_{Nw}*,* q_{SG}*, resp.* q_{C} *total queries to their* NEW*,* SIGN*, resp.* CORRUPT *oracle, and making at most* $q_{\mathrm{SG/U}}$ *queries* SIGN$(i, \cdot)$ *for any user* i*.*

Multi-user EUF-CMA security of signature schemes (with adaptive corruptions) can be reduced to classical, single-user EUF-CMA security (formally, $\mathrm{G}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{S},\mathcal{A}}$ with $\mathcal{A}$ restricted to $q_{\mathrm{Nw}} = 1$ queries to NEW) by a standard hybrid argument, losing a factor of number of users. Formally, this yields

$$\mathsf{Adv}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{S}}(t, q_{\mathrm{Nw}}, q_{\mathrm{SG}}, q_{\mathrm{SG/U}}, q_{\mathrm{C}}) \leq q_{\mathrm{Nw}} \cdot \mathsf{Adv}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{S}}(t', 1, q_{\mathrm{SG/U}}, q_{\mathrm{SG/U}}, 0),$$

where $t \approx t'$. (Note that the reduction is not affected by the total number of signature queries q_{SG} across all users.) In many cases, such loss is indeed unavoidable [**?**].

Definition 3.7 (MAC scheme)**.** *A* MAC scheme $\mathsf{M} = (\mathsf{KGen}, \mathsf{Tag}, \mathsf{Vrfy})$ *consists of three efficient algorithms defined as follows.*

- $\mathsf{KGen}() \xrightarrow{\$} K$*. This probabilistic algorithm generates a key K.*
- $\mathsf{Tag}(K, m) \xrightarrow{\$} \tau$*. On input a key K and a message m, this (possibly) probabilistic algorithm outputs a message authentication code (MAC) τ.*

$\mathrm{G}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{S},\mathcal{A}}$

```
INITIALIZE:
    $Q \leftarrow \emptyset$
    $\mathcal{C} \leftarrow \emptyset$
    $u \leftarrow 0$

CORRUPT($i$):
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{i\}$
    return $sk_i$

NEW:
    $u \leftarrow u + 1$
    $(pk_u, sk_u) \xleftarrow{\$} \mathsf{KGen}()$
    return $pk_u$

SIGN($i, m$):
    $\sigma \xleftarrow{\$} \mathsf{Sign}(sk_i, m)$
    $Q \leftarrow Q \cup \{(i, m)\}$
    return $\sigma$

FINALIZE($i^*, m^*, \sigma^*$):
    $d^* \leftarrow \mathsf{Vrfy}(pk_{i^*}, m^*, \sigma^*)$
    return $[[d^* = 1 \wedge i^* \notin \mathcal{C} \wedge (i^*, m^*) \notin Q]]$
```

$\mathrm{G}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{M},\mathcal{A}}$

```
INITIALIZE:
    $Q \leftarrow \emptyset$
    $\mathcal{C} \leftarrow \emptyset$
    $u \leftarrow 0$

CORRUPT($i$):
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{i\}$
    return $K_i$

NEW:
    $u \leftarrow u + 1$
    $K_u \xleftarrow{\$} \mathsf{KGen}()$

TAG($i, m$):
    $\tau \xleftarrow{\$} \mathsf{Tag}(K_i, m)$
    $Q \leftarrow Q \cup \{(i, m)\}$
    return $\tau$

VRFY($i, m, \tau$):
    $d \leftarrow \mathsf{Vrfy}(K_i, m, \tau)$
    return $d$

FINALIZE($i^*, m^*, \tau^*$):
    $d^* \leftarrow \mathsf{Vrfy}(K_{i^*}, m^*, \tau^*)$
    return $[[d^* = 1 \wedge i^* \notin \mathcal{C} \wedge (i^*, m^*) \notin Q]]$
```

Figure 3: Multi-user existential unforgeability ($\mathsf{mu\text{-}EUF\text{-}CMA}$) of signature schemes (top) and MAC schemes (bottom).

- $\mathsf{Vrfy}(K, m, \tau) \rightarrow d$. *On input a key K, a message m, and a MAC τ, this deterministic algorithm outputs a decision bit $d \in \{0, 1\}$ (where $d = 1$ indicates validity of the MAC).*

Definition 3.8 (MAC $\mathsf{mu\text{-}EUF\text{-}CMA}$ security)**.** *Let M be a MAC scheme and $\mathrm{G}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{M},\mathcal{A}}$ be the game for MAC multi-user existential unforgeability under chosen-message attacks with adaptive corruptions firstoftwodefined as in Figure 3(see the full version [**?**] for the formal definition). We define*

$$\mathsf{Adv}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{M}}(t, q_{\mathrm{Nw}}, q_{\mathrm{TG}}, q_{\mathrm{TG/U}}, q_{\mathrm{VF}}, q_{\mathrm{VF/U}}, q_{\mathrm{C}}) := \max_{\mathcal{A}} \Pr\left[\mathrm{G}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{M},\mathcal{A}} \Rightarrow 1\right],$$

where the maximum is taken over all adversaries, denoted $(t, q_{\mathrm{Nw}}, q_{\mathrm{TG}}, q_{\mathrm{TG/U}}, q_{\mathrm{VF}}, q_{\mathrm{VF/U}}, q_{\mathrm{C}})$-firstofone $\mathsf{mu\text{-}EUF\text{-}CMA}$-adversaries*, running in time at most t and making at most q_{Nw}, q_{TG}, q_{VF}, resp. q_{C} queries to their* NEW*,* SIGN*,* VRFY*, resp.* CORRUPT *oracle, and making at most $q_{\mathrm{TG/U}}$ queries* TAG$(i, \cdot)$*, resp. $q_{\mathrm{VF/U}}$ queries* VRFY$(i, \cdot)$ *for any user i.*

As for signature schemes, multi-user EUF-CMA security of MACs reduces to the single-user case ($q_{\mathrm{Nw}} = 1$) by a standard hybrid argument:

$$\begin{aligned}&\mathsf{Adv}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{M}}(t, q_{\mathrm{Nw}}, q_{\mathrm{TG}}, q_{\mathrm{TG/U}}, q_{\mathrm{VF}}, q_{\mathrm{VF/U}}, q_{\mathrm{C}})\\ &\leq q_{\mathrm{Nw}} \cdot \mathsf{Adv}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{M}}(t, 1, q_{\mathrm{TG/U}}, q_{\mathrm{TG/U}}, q_{\mathrm{VF/U}}, q_{\mathrm{VF/U}}, 0),\end{aligned}$$

where $t \approx t'$. (Note that the reduction is not affected by the total number of tagging and verification queries q_{TG} resp. q_{VF} across all users.)

Our multi-user definition of MACs provides a verification oracle, which is non-standard (and in general not equivalent to a definition with a single forgery attempts, as Bellare, Goldreich and

Mityiagin [?] showed). For PRF-based MACs (which in particular includes HMAC used in TLS 1.3), it however is equivalent and the reduction from multi-query to single-query verification is tight [?].

In our key exchange reductions, we actually do not need to corrupt MAC keys, i.e., we achieve $q_{\mathrm{C}} = 0$. This in particular allows specific constructions like AMAC [?] achieving tight multi-user security (without corruptions).

If we use a random oracle RO as PRF-like MAC with key length kl and output length ol, then

$$\mathsf{Adv}_{\mathsf{RO}}^{\mathsf{mu\text{-}EUF\text{-}CMA}}(t, q_{\mathrm{Nw}}, q_{\mathrm{Tg}}, q_{\mathrm{Tg/U}}, q_{\mathrm{Vf}}, q_{\mathrm{Vf/U}}, q_{\mathrm{C}}, q_{\mathrm{RO}}) \leq \frac{q_{\mathrm{Vf}}}{2^{ol}} + \frac{(q_{\mathrm{Nw}} - q_{\mathrm{C}}) \cdot q_{\mathrm{RO}}}{2^{kl}}.$$

3.4 Hash Function Collision Resistance

Finally, let us define collision resistance of hash functions.

Definition 3.9 (Hash function collision resistance)**.** *Let* $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^{ol}$ *for* $ol \in \mathbb{N}$ *be a function. For a given adversary* $\mathcal{A}$ *running in time at most* t*, we can consider*

$$\mathsf{Adv}_{\mathsf{H}}^{\mathsf{CR}}(t) := \Pr\left[(m, m') \xleftarrow{\$} \mathcal{A} : m \neq m' \text{ and } \mathsf{H}(m) = \mathsf{H}(m')\right].$$

If we use a random oracle RO as hash function, thenfirstofone by the birthday bound

$$\mathsf{Adv}_{\mathsf{RO}}^{\mathsf{CR}}(t, q_{\mathrm{RO}}) \leq \frac{q_{\mathrm{RO}}^2}{2^{ol+1}} + \frac{1}{2^{ol}}.$$

4 Proof of the Strong Diffie–Hellman GGM Bound (Theorem 3.3)

We begin by giving a code-based game for the strong Diffie–Hellman problem in the generic group model. First, we establish some preliminaries, using the setting and notation of Bellare and Dai [?]. Let $\mathbb{G}$ be an arbitrary set of strings with prime order p, and let $E : \mathbb{Z}_p \to \mathbb{G}$ be a bijection, called the encoding function. For any two strings $A, B \in \mathbb{G}$, we define the operation $A \,\mathrm{OP}_E\, B = E(E^{-1}(A) + E^{-1}(B) \mod p)$. The set $\mathbb{G}$ is a group with respect to this operation, and it is isomorphic to $\mathbb{Z}_p$. Therefore, $\mathbb{G}$ has the identity $E(0)$, and it is generated by $E(1)$.

In the generic group model, we wish for the adversary to compute group operations only through an oracle OP. We accomplish this by picking the encoding function E at random and keeping it secret; then providing oracle access to OP_E through OP. In this model, we can give a sequence of games bounding the advantage of any adversary $\mathcal{A}$ that makes t queries to the OP oracle and q queries to the stDH oracle.

Game 0. This first game formalizes the strong Diffie–Hellman problem in the generic group model. Note that for any $a \in \mathbb{Z}_p$, a is the discrete logarithm of the group element $E(a)$.

It follows that

$$\mathsf{Adv}_{\mathbb{G}}^{\mathsf{stDH}}(t, q_{\mathrm{sDH}}) = \Pr[\mathrm{G}_0 \Rightarrow 1].$$

Game 1. In Game G_1, we change the internal notation of the game. First, for clarity and without loss of generality, we assume the adversary queries its OP and stDH oracles only on valid inputs (meaning their inputs are valid group elements in GL). Instead of representing each element of $\mathbb{G}$ with an element of $\mathbb{Z}_p$, we use a vector over $\mathbb{Z}_p^3$. We define the basis vectors $\vec{e_1} := (1,0,0)$, $\vec{e_2} := (0,1,0)$, and $\vec{e_3} := (0,0,1)$. We map $\mathbb{Z}_p^3$ to $\mathbb{Z}_p$ by taking the inner product with the vector $(1, x, y)$. (Effectively, we are representing each element of $\mathbb{Z}_p$ as a linear combination modulo p of

```
G_0
Initialize():
  $p \leftarrow |\mathbb{G}|$; $E \xleftarrow{\$}$ Bijections$(\mathbb{Z}_p, \mathbb{G})$
  $\mathbb{1} \leftarrow E(0)$; $g \leftarrow E(1)$
  $x, y \xleftarrow{\$} \mathbb{Z}_p^*$; $X \leftarrow E(x)$; $Y \leftarrow E(y)$
  $GL \leftarrow \{\mathbb{1}, g, x, y\}$
  return $(\mathbb{1}, g, x, y)$

OP(A, B, sgn ):
  if $A \notin GL$ or $B \notin GL$ then return $\perp$
  $c \leftarrow E^{-1}(A)$ sgn $E^{-1}(B) \mod p$
  $C \leftarrow E(c)$; $GL \leftarrow GL \cup \{C\}$
  return $C$

stDH(A, B):
  if $A \notin GL$ or $B \notin GL$ then return $\perp$
  $z \leftarrow x \cdot E^{-1}(A) \mod p$
  $Z \leftarrow E(z)$
  return $[[Z = B]]$

Finalize(Z):
  if $Z \notin GL$ then return false
  $z \leftarrow x \cdot y \mod p$; return $[[Z = E(z)]]$
```

Figure 4: Game G_0 of the stDH proof.

1, x, and y.) We cache the map from $\mathbb{Z}_p^3$ to $\mathbb{G}$ induced by this transformation in a table TV and its inverse map in a table TI.

Although one element of $\mathbb{G}$ may now have multiple representations, the bilinearity of the inner product ensures that the view of the adversary is not changed, and $\Pr[G_1] = \Pr[G_0]$.

Game 2. Next, we replace the random encoding function E with a lazily sampled encoding represented by table TV for the forward direction and TI for the backward direction. Because we want our encoding to be one-to-one, we sample from the set $\mathbb{G} \setminus GL$. This assigns a unique element of $\mathbb{G}$ to each vector $\vec{t}$. However, as we've noted, each integer in $\mathbb{Z}_p$ has multiple representations in $\mathbb{Z}_p^3$. If two representations of the same integer are submitted to the encoding algorithm VE, we set a bad flag and program the encoding table to maintain consistency.

We also change the format of the check in the stDH oracle. Since $\mathrm{VE}(x\vec{a}) = B = \mathrm{VE}(\vec{b})$ if and only if $\langle x\vec{a}, \vec{x}\rangle = \langle \vec{b}, \vec{x}\rangle$, we return true if the latter condition holds and false otherwise. These two conditions are equivalent, so $\Pr[G_2] = \Pr[G_1]$.

Game 3. In this game, we stop programming the encoding table after the bad flag is set. Let F_1 denote the event that G_3 sets the bad flag at any point. By the fundamental lemma of game playing, $\Pr[G_2] \leq \Pr[G_3 \text{ and } \overline{F_1} + \Pr[F_1]$.

Game 4. We remove the now-redundant bad flag, but the Finalize oracle now returns true if at any point in game G_3 the bad flag would have been set (i.e. if event F_1 occurs). Otherwise, all oracles behave exactly as they did in G_3. It follows that $\Pr[G_3 \text{ and } \overline{F_1}] + \Pr[F_1] \leq \Pr[G_4]$.

Additionally, in the stDH oracle, we separate out checking for trivial queries: if the adversary computed $A = g^a$ and $B = X^a$ for an integer a of their choosing. If this is so, then $\vec{a} = a\vec{e_1}$ and $\vec{b} = a\vec{e_2}$, so $\langle x\vec{a}, \vec{x}\rangle = xa = \langle \vec{b}, \vec{x}\rangle$, so may return true. If the query is nontrivial but should still return true according to our previous condition, we set a bad_2 flag. This does not change the oracle's response to any query, so the above bound still holds.

Game 5. In Game G_5, we no longer return true in the stDH oracle after the bad_2 flag is set. This makes the second check redundant and has the effect that the stDH oracle's behavior is no longer dependent on the value of either x or $\vec{x}$. Let event F_2 denote the event that G_5 sets the bad_2 flag. By the fundamental lemma of game playing, $\Pr[G_4] \leq \Pr[G_5 \text{ and } \overline{F_2}] + \Pr[F_2]$.

```
$\underline{\mathrm{G}_1}$

INITIALIZE():
  $p \leftarrow |\mathbb{G}|$; $E \xleftarrow{\$}$ Bijections($\mathbb{Z}_p, \mathbb{G}$)
  $k \leftarrow 0$; $\mathbb{1} \leftarrow \mathrm{VE}(\vec{0})$; $g \leftarrow \mathrm{VE}(\vec{e_1})$
  $x, y \xleftarrow{\$} \mathbb{Z}_p^*$; $\vec{x} \leftarrow \mathbb{1}, x, y$
  $X \leftarrow \mathrm{VE}(\vec{e_2})$; $Y \leftarrow \mathrm{VE}(\vec{e_3})$
  return $(\mathbb{1}, g, x, y)$

OP($A, B$, sgn ):
  $\vec{c} \leftarrow \mathrm{VE}^{-1}(A)$ sgn $\mathrm{VE}^{-1}(B)$ mod $p$
  $C \leftarrow \mathrm{VE}(\vec{c})$; return $C$

VE($\vec{t}$):
  if $TV[\vec{t}] \neq \perp$ then return $TV[\vec{t}]$
  $k \leftarrow k + 1$; $\vec{t_k} \leftarrow \vec{t}$
  $v \leftarrow \langle \vec{t}, \vec{x} \rangle$; $C \leftarrow E(v)$; $GL \leftarrow GL \cup \{C\}$
  $TV[\vec{t}] \leftarrow C$; $TI[C] \leftarrow \vec{t}$
  return $TV[\vec{t}]$

stDH($A, B$):
  $\vec{a} \leftarrow \mathrm{VE}^{-1}(A)$; $\vec{b} \leftarrow \mathrm{VE}^{-1}(B)$
  return $[[\mathrm{VE}(x\vec{a}) = B]]$

FINALIZE($Z$):
  return $[[\mathrm{VE}(x\vec{e_3}) = Z]]$

VE$^{-1}$($C$):
  return $TI[C]$
```

Figure 5: Game G_1 of the stDH proof.

Game 6. In Game G_6, we remove the redundant check and bad flag from the stDH oracle, and in the FINALIZE oracle we return true whenever the bad_2 flag would have been set in G_5. Otherwise all oracles behave precisely as they did in G_5. It follows that $\Pr[\mathrm{G}_5 \text{ and } \overline{F_2}] + \Pr[F_2] \leq \Pr[\mathrm{G}_6]$. We also move the initialization of variables x, y, and $\vec{x}$ from INITIALIZE to FINALIZE. Since these variables are not used by any oracle but FINALIZE, this does not change the view of the adversary.

At this point, we can collect the bounds from each gamehop to see that

$$\mathsf{Adv}_{\mathbb{G}}^{\mathsf{stDH}}(t, q_{\mathrm{sDH}}) \leq \Pr[\mathrm{G}_6].$$

Therefore we analyze the advantage of an adversary in game G_6.

We can separately analyze each condition of FINALIZE. We know that x and y are sampled independently of the $t + 4$ entries of TV. For each index $i \in [1 \ldots t + 4]$, let F_i be the bivariate linear polynomial over $\mathbb{Z}_p$ whose coefficients are given by the vector $\vec{t_i}$. Then for any pair of vectors $(\vec{t_i}, \vec{t_j})$, the condition $\langle \vec{t_i} - \vec{t_j} \rangle = 0$ holds only if $(1, x, y)$ is a root of $F_i - F_j$. Using Lemma 1 of [**?**] and a union bound over all pairs, the probability of this event is at most $(t + 4)^2/p$.

For the second condition; we see that for any $(\vec{t_i}, \vec{t_j})$, it is true that $\langle x\vec{t_i} - \vec{t_j} \rangle = 0$ only if $(1, x, y)$ is a root of $XF_i - F_j$, which is a bivariate quadratic polynomial over $\mathbb{Z}_p$. Again Using Lemma 1 and a union bound, this occurs with probability at most $2(t + 4)^2/p$.

If neither event occurs, then the adversary wins only if $[\mathrm{VE}(x\vec{e_3}) = Z]$. Because the second condition failed, we know that $(x\vec{e_3})$ is not an entry in table TV. Therefore the response to $\mathrm{VE}(x\vec{e_3})$ will be sampled uniformly at random, and it will equal Z with probability $1/p$. Then by the union bound, $\Pr[\mathrm{G}_6] \leq (3(t + 4)^2 + 1)/p$. Collecting the bounds gives the theorem statement for all $t > 25$. □

5 The SIGMA Protocol

The SIGMA family of key exchange protocols introduced by Krawczyk [**?**, **?**] describes several variants for building authenticated Diffie–Hellman key exchange using the "SIGn-and-MAc" approach.

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

```
stDH$(A, B)$:
  $\vec{a} \leftarrow \mathrm{VE}^{-1}(A)$; $\vec{b} \leftarrow \mathrm{VE}^{-1}(B)$
  if $\langle x\vec{a}, \vec{x}\rangle = \langle \vec{b}, \vec{x}\rangle$ then return true
  return false

VE$(\vec{t})$:
  if $TV[\vec{t}] \neq \bot$ then return $TV[\vec{t}]$
  $C \leftarrow \mathbb{G} \setminus GL$
  if $(\exists \vec{s} : TV[\vec{s}] \neq \bot$ and $\langle \vec{t}, \vec{x}\rangle = \langle \vec{s}, \vec{x}\rangle)$
     then bad $\leftarrow$ true; $\boxed{C \leftarrow TV[\vec{s}]}$
  $k \leftarrow k + 1$; $\vec{t_k} \leftarrow \vec{t}$
  $GL \leftarrow GL \cup \{C\}$
  $TV[\vec{t}] \leftarrow C$; $TI[C] \leftarrow \vec{t}$
  return $TV[\vec{t}]$
```

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

```
stDH$(A, B)$:
  $\vec{a} \leftarrow \mathrm{VE}^{-1}(A)$; $\vec{b} \leftarrow \mathrm{VE}^{-1}(B)$; $a \leftarrow \vec{a}[1]$
  if $\vec{a} = a\vec{e_1}$ and $\vec{b} = a\vec{e_2}$ then return true
  if $\langle x\vec{a}, \vec{x}\rangle = \langle \vec{b}, \vec{x}\rangle$ then bad$_2$ $\leftarrow$ true; $\boxed{\text{return true}}$
  return false

VE$(\vec{t})$:
  if $TV[\vec{t}] \neq \bot$ then return $TV[\vec{t}]$
  $C \leftarrow \mathbb{G} \setminus GL$
  $k \leftarrow k + 1$; $\vec{t_k} \leftarrow \vec{t}$
  $GL \leftarrow GL \cup \{C\}$
  $TV[\vec{t}] \leftarrow C$; $TI[C] \leftarrow \vec{t}$
  return $TV[\vec{t}]$

FINALIZE$(Z)$:
  if $\exists i, j : 1 \leq i < j \leq k$ and $\langle \vec{t_i} - \vec{t_j}, \vec{x}\rangle = 0$
     then return true
  return $[[\mathrm{VE}(x\vec{e_3}) = Z]]$
```

G_6

```
INITIALIZE():
  $p \leftarrow |\mathbb{G}|$;
  $k \leftarrow 0; \mathbb{1} \leftarrow \mathrm{VE}(\vec{0})$; $g \leftarrow \mathrm{VE}(\vec{e_1})$
  $X \leftarrow \mathrm{VE}(\vec{e_2})$; $Y \leftarrow \mathrm{VE}(\vec{e_3})$
  return $(\mathbb{1}, g, x, y)$

FINALIZE$(Z)$:
  $x, y \xleftarrow{\$} \mathbb{Z}_p^*$; $\vec{x} \leftarrow (\mathbb{1}, x, y)$
  if $\exists i, j : 1 \leq i < j \leq k$ and $\langle \vec{t_i} - \vec{t_j}, \vec{x}\rangle = 0)$
     then return true
  if $\exists i, j : 1 \leq i < j \leq k$
  and $\langle x\vec{t_i} - \vec{t_j}, \vec{x}\rangle = 0$ or $\langle x\vec{t_j} - \vec{t_i}, \vec{x}\rangle 0$
     then return true
  return $[[\mathrm{VE}(x\vec{e_3}) = Z]]$

stDH$(A, B)$:
  $\vec{a} \leftarrow \mathrm{VE}^{-1}(A)$; $\vec{b} \leftarrow \mathrm{VE}^{-1}(B)$;$a \leftarrow \vec{a}[1]$
  if $(\vec{a} = a\vec{e_1}$ and $\vec{b} = a\vec{e_2})$ then return true
  return 0
```

Figure 6: Top left: Games G_2 (changes highlighted in gray) and G_3 (changes highlighted in frames) of the strong Diffie–Hellman proof. Top right: Games G_4 and G_5. Bottom: Game G_6 (changes highlighted in gray) of the strong Diffie–Hellman proof.

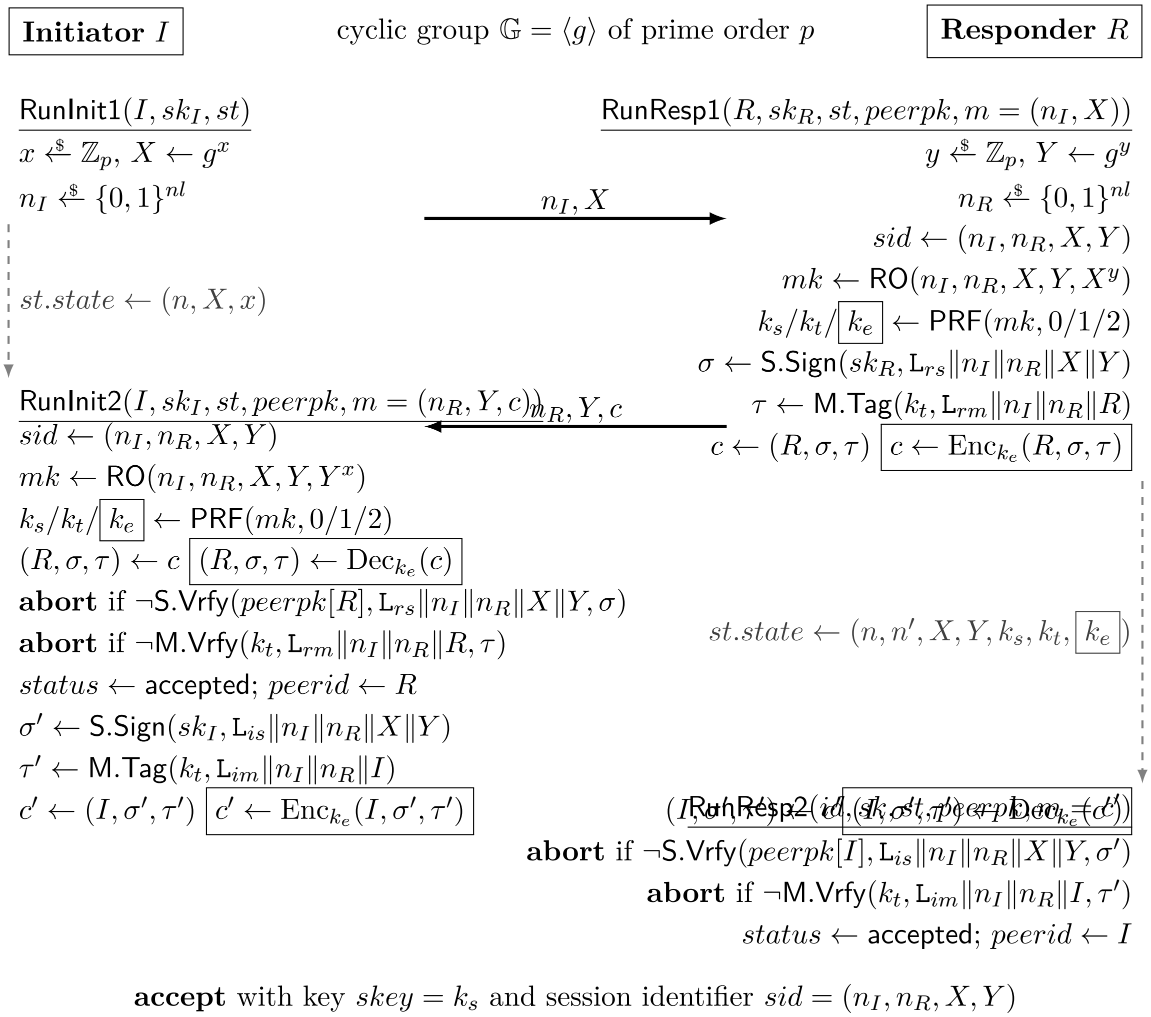

Initiator I
cyclic group $\mathbb{G} = \langle g \rangle$ of prime order p
Responder R
$\underline{\mathsf{RunInit1}(I, sk_I, st)}$
$x \xleftarrow{\$} \mathbb{Z}_p,\ X \leftarrow g^x$
$n_I \xleftarrow{\$} \{0,1\}^{nl}$
$st.state \leftarrow (n, X, x)$
n_I, X
$\underline{\mathsf{RunResp1}(R, sk_R, st, peerpk, m = (n_I, X))}$
$y \xleftarrow{\$} \mathbb{Z}_p,\ Y \leftarrow g^y$
$n_R \xleftarrow{\$} \{0,1\}^{nl}$
$sid \leftarrow (n_I, n_R, X, Y)$
$mk \leftarrow \mathsf{RO}(n_I, n_R, X, Y, X^y)$
$k_s/k_t/\boxed{k_e} \leftarrow \mathsf{PRF}(mk, 0/1/2)$
$\sigma \leftarrow \mathsf{S.Sign}(sk_R, \mathsf{L}_{rs} \| n_I \| n_R \| X \| Y)$
$\tau \leftarrow \mathsf{M.Tag}(k_t, \mathsf{L}_{rm} \| n_I \| n_R \| R)$
$c \leftarrow (R, \sigma, \tau)\ \boxed{c \leftarrow \mathrm{Enc}_{k_e}(R, \sigma, \tau)}$
n_R, Y, c
$\underline{\mathsf{RunInit2}(I, sk_I, st, peerpk, m = (n_R, Y, c))}$
$sid \leftarrow (n_I, n_R, X, Y)$
$mk \leftarrow \mathsf{RO}(n_I, n_R, X, Y, Y^x)$
$k_s/k_t/\boxed{k_e} \leftarrow \mathsf{PRF}(mk, 0/1/2)$
$(R, \sigma, \tau) \leftarrow c\ \boxed{(R, \sigma, \tau) \leftarrow \mathrm{Dec}_{k_e}(c)}$
abort if $\neg\mathsf{S.Vrfy}(peerpk[R], \mathsf{L}_{rs} \| n_I \| n_R \| X \| Y, \sigma)$
abort if $\neg\mathsf{M.Vrfy}(k_t, \mathsf{L}_{rm} \| n_I \| n_R \| R, \tau)$
$status \leftarrow \mathsf{accepted};\ peerid \leftarrow R$
$\sigma' \leftarrow \mathsf{S.Sign}(sk_I, \mathsf{L}_{is} \| n_I \| n_R \| X \| Y)$
$\tau' \leftarrow \mathsf{M.Tag}(k_t, \mathsf{L}_{im} \| n_I \| n_R \| I)$
$c' \leftarrow (I, \sigma', \tau')\ \boxed{c' \leftarrow \mathrm{Enc}_{k_e}(I, \sigma', \tau')}$
$st.state \leftarrow (n, n', X, Y, k_s, k_t, \boxed{k_e})$
$\underline{\mathsf{RunResp2}(id, sk, st, peerpk, m = c')}$
$(I, \sigma', \tau') \leftarrow c'\ \boxed{(I, \sigma', \tau') \leftarrow \mathrm{Dec}_{k_e}(c')}$
abort if $\neg\mathsf{S.Vrfy}(peerpk[I], \mathsf{L}_{is} \| n_I \| n_R \| X \| Y, \sigma')$
abort if $\neg\mathsf{M.Vrfy}(k_t, \mathsf{L}_{im} \| n_I \| n_R \| I, \tau')$
$status \leftarrow \mathsf{accepted};\ peerid \leftarrow I$
accept with key $skey = k_s$ and session identifier $sid = (n_I, n_R, X, Y)$

Figure 7: The SIGMA/SIGMA-I protocol flow diagram. $\boxed{\text{Boxed}}$ code is only performed in the SIGMA-I variant. Values L_x indicate label strings (distinct per x).

Its design has been adopted in several Internet security protocols, including, e.g., the Internet Key Exchange protocol [?, ?] as part of the IPsec Internet security protocol [?] and the newest version 1.3 of the Transport Layer Security (TLS) protocol [?].

Beyond the basic SIGMA design, we are particularly interested in the SIGMA-I variant which forms the basis of the TLS 1.3 key exchange and aims at hiding the protocol participants' identities as additional feature. We here present an augmented version of the basic SIGMA/SIGMA-I protocols which includes explicit exchange of session-identifying random numbers (nonces) to be closer to SIGMA(-like) protocols in practice, somewhat following the "full-fledged" SIGMA variant [?, Appendix B]. We illustrate these protocol flows in Figure 7. and Figure 8 formalizes both as key exchange protocols according to the syntax of Section 2.1.

The SIGMA and SIGMA-I protocols make use of a signature scheme $\mathsf{S} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$, a MAC scheme $\mathsf{M} = (\mathsf{KGen}, \mathsf{Tag}, \mathsf{Vrfy})$, a pseudorandom function PRF, and a function RO which we model as a random oracle. The parties' long-term secret keys consist of one signing key, i.e., $\mathsf{KE.KGen} = \mathsf{S.KGen}$. The protocols consists of three messages exchanged and accordingly two steps performed by both initiator and responder, which we describe in more detail now.

```
$\mathsf{Activate}(id, sk, peerid, peerpk, role)$:
$st'.role \leftarrow role$
$st'.status \leftarrow$ running
if $role =$ initiator then
    $(st', m') \leftarrow \mathsf{RunInit1}(id, sk, st')$
else $m' \leftarrow \bot$
return $(st', m')$
```

```
$\mathsf{Run}(id, sk, st, peerpk, m)$:
if $st.status \neq$ running then
return $\bot$
if $st.role =$ initiator then
    $(st', m') \leftarrow \mathsf{RunInit2}(id, sk, st, peerpk, m)$
else if $st.sid = \bot$
    $(st', m') \leftarrow \mathsf{RunResp1}(id, sk, st, peerpk, m)$
else
    $(st', m') \leftarrow \mathsf{RunResp2}(id, sk, st, peerpk, m)$
return $(st', m')$
```

```
$\mathsf{RunInit1}(id, sk, st)$:
$n_I \xleftarrow{\$} \{0,1\}^{nl}$
$x \xleftarrow{\$} \mathbb{Z}_p$
$X \leftarrow g^x$
$st'.state \leftarrow (n_I, X, x)$
$m' \leftarrow (n_I, X)$
return $(st', m')$
```

```
$\mathsf{RunResp1}(id, sk, st, peerpk, m)$:
$(n_I, X) \leftarrow m$
$n_R \xleftarrow{\$} \{0,1\}^{nl}$
$y \xleftarrow{\$} \mathbb{Z}_p$
$Y \leftarrow g^y$
$st'.sid \leftarrow (n_I, n_R, X, Y)$
$\sigma \leftarrow \mathsf{S.Sign}(sk, \mathsf{L}_{rs} \| n_I \| n_R \| X \| Y)$
$mk \leftarrow \mathsf{RO}(n_I \| n_R \| X \| Y \| X^y)$
$k_s \leftarrow \mathsf{PRF}(mk, 0)$
$k_t \leftarrow \mathsf{PRF}(mk, 1)$
$\boxed{k_e \leftarrow \mathsf{PRF}(mk, 2)}$
$\tau \leftarrow \mathsf{M.Tag}(k_t, \mathsf{L}_{rm} \| n_I \| n_R \| id)$
$st'.state \leftarrow (n_I, n_R, X, Y, k_s, k_t)$
$\boxed{st'.state \leftarrow (n_I, n_R, X, Y, k_s, k_t, k_e)}$
$m' \leftarrow (n_R, Y, id, \sigma, \tau)$
$\boxed{m' \leftarrow (n_R, Y, \mathsf{Enc}(k_e, (id, \sigma, \tau)))}$
return $(st', m')$
```

```
$\mathsf{RunInit2}(id, sk, st, peerpk, m)$:
$(n_R, Y, peerid, \sigma, \tau) \leftarrow m$
$\boxed{(n_R, Y, c) \leftarrow m}$
$(n_I, X, x) \leftarrow st.state$
$st'.sid \leftarrow (n_I, n_R, X, Y)$
$mk \leftarrow \mathsf{RO}(n_I \| n_R \| X \| Y \| Y^x)$
$k_s \leftarrow \mathsf{PRF}(mk, 0)$
$k_t \leftarrow \mathsf{PRF}(mk, 1)$
$\boxed{k_e \leftarrow \mathsf{PRF}(mk, 2)}$
$\boxed{(peerid, \sigma, \tau) \leftarrow \mathsf{Dec}(k_e, c)}$
$st'.peerid \leftarrow peerid$
if $\mathsf{S.Vrfy}(peerpk[peerid], \mathsf{L}_{rs} \| n_I \| n_R \| X \| Y, \sigma)$
and $\mathsf{M.Vrfy}(k_t, \mathsf{L}_{rm} \| n_I \| n_R \| peerid, \tau)$ then
    $st'.status \leftarrow$ accepted
    $st'.skey \leftarrow k_s$
    $\sigma' \leftarrow \mathsf{S.Sign}(sk, \mathsf{L}_{is} \| n_I \| n_R \| X \| Y)$
    $\tau' \leftarrow \mathsf{M.Tag}(k_t, \mathsf{L}_{im} \| n_I \| n_R \| id)$
    $m' \leftarrow (id, \sigma', \tau')$
    $\boxed{m' \leftarrow \mathsf{Enc}(k_e, (id, \sigma', \tau'))}$
else
    $m' \leftarrow \bot$
    $st'.status \leftarrow$ rejected
return $(st', m')$
```

```
$\mathsf{RunResp2}(id, sk, st, peerpk, m)$:
$(n_I, n_R, X, Y, k_s, k_t) \leftarrow st.state$
$\boxed{(n_I, n_R, X, Y, k_s, k_t, k_e) \leftarrow st.state}$
$(peerid, \sigma', \tau') \leftarrow m$
$\boxed{(peerid, \sigma', \tau') \leftarrow \mathsf{Dec}(k_e, m)}$
$st'.peerid \leftarrow peerid$
if $\mathsf{S.Vrfy}(peerpk[peerid], \mathsf{L}_{is} \| n_I \| n_R \| X \| Y, \sigma')$
and $\mathsf{M.Vrfy}(k_t, \mathsf{L}_{im} \| n_I \| n_R \| peerid, \tau')$ then
    $st'.status \leftarrow$ accepted
    $st'.skey \leftarrow k_s$
else $st'.status \leftarrow$ rejected
$m' \leftarrow \varepsilon$
return $(st', m')$
```

Figure 8: The formalized SIGMA/SIGMA-I key exchange protocols (cf. Section 2.1). $\boxed{\text{Boxed}}$ code is only performed in the SIGMA-I variant.

Initiator Step 1. The initiator picks a Diffie–Hellman exponent $x \xleftarrow{\$} \mathbb{Z}_p$ and a random nonce n_I of length nl and sends n_I and g^x.

Responder Step 1. The responder also picks a random DH exponent y and a random nonce n_R. It then derives a master key as $mk \leftarrow \mathsf{RO}(n_I, n_R, X, Y, X^y)$ from nonces, DH shares, and the joint DH secret $g^{xy} = (g^x)^y$. From mk, keys are derived via PRF with distinct labels: the session key k_s, the MAC key k_t, and (only in SIGMA-I) the encryption key k_e.

The responder computes a signature σ with sk_R over nonces and DH shares (and a unique label $\mathtt{L}_{rs}$) and a MAC value τ under key k_t over the nonces and its identity R (and unique label $\mathtt{L}_{rm}$). It sends n_I, g^y, as well as R, σ, and τ to the initiator. In SIGMA-I the last three elements are encrypted using k_e to conceal the responder's identity against passive adversaries.

Initiator Step 2. The initiator also computes mk and keys k_s, k_t, and (in SIGMA-I, used to decrypt the second message part) k_e. It ensures both the received signature σ and MAC τ verify, and aborts otherwise.

It computes its own signature σ' under sk_I on nonces and DH shares (with a different label $\mathtt{L}_{is}$) and a MAC τ' under k_t over the nonces and its identity I (with yet another label $\mathtt{L}_{im}$). It sends I, σ', and τ' to the responder (in SIGMA-I encrypted under k_e) and accepts with session key k_s using the nonces and DH shares (n_I, n_R, X, Y) as session identifier.

Responder Step 2. The responder finally checks the initiator's signature σ' and MAC τ' (aborting if either fails) and then accepts with session key $skey = k_s$ and session identifier $sid = (n_I, n_R, X, Y)$.

6 Tighter Security Proof for SIGMA-I

We now come to our first main result, a tighter security proof for the SIGMA-I protocol. Note that by omitting message encryption our proof similarly applies to the basic SIGMA protocol.

Theorem 6.1. *Let the SIGMA-I protocol be as specified in Figure firstoftwo87 based on a group $\mathbb{G}$ of prime order p, a PRF PRF, a signature scheme S, and a MAC M, and let RO in the protocol be modeled as a random oracle. For any $(t, q_{\mathrm{N}}, q_{\mathrm{S}}, q_{\mathrm{RS}}, q_{\mathrm{RL}}, q_{\mathrm{T}})$-$\mathsf{KE\text{-}SEC}$-adversary against SIGMA-I making at most q_{RO} queries to RO, we give algorithms $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{B}_3$, and $\mathcal{B}_4$ in the proof, with running times $t_{\mathcal{B}_1} \approx t + 2q_{\mathrm{RO}} \log_2 p$ and $t_{\mathcal{B}_i} \approx t$ (for $i = 2, \dots, 4$) close to that of $\mathcal{A}$, such that*

$$\begin{aligned}
&\mathsf{Adv}^{\mathsf{KE\text{-}SEC}}_{\text{SIGMA-I}}(t, q_{\mathrm{N}}, q_{\mathrm{S}}, q_{\mathrm{RS}}, q_{\mathrm{RL}}, q_{\mathrm{T}}) \\
&\quad \le \frac{3q_{\mathrm{S}}^2}{2^{nl+1} \cdot p} + \mathsf{Adv}^{\mathsf{stDH}}_{\mathbb{G}}(t_{\mathcal{B}_1}, q_{\mathrm{RO}}) + \mathsf{Adv}^{\mathsf{mu\text{-}PRF}}_{\mathsf{PRF}}(t_{\mathcal{B}_2}, q_{\mathrm{S}}, 3q_{\mathrm{S}}, 3) \\
&\qquad + \mathsf{Adv}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{S}}(t_{\mathcal{B}_3}, q_{\mathrm{N}}, q_{\mathrm{S}}, q_{\mathrm{S}}, q_{\mathrm{RL}}) + \mathsf{Adv}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{M}}(t_{\mathcal{B}_4}, q_{\mathrm{S}}, q_{\mathrm{S}}, 1, q_{\mathrm{S}}, 1, 0).
\end{aligned}$$

Here, nl is the nonce length in SIGMA-I and $\mathbb{G}$ is the used Diffie–Hellman group of prime order p.

In terms of multi-user security for the employed primitives, multi-user PRF and MAC security can be obtained tightly, e.g., via the efficient AMAC construction [**?**], and multi-user signature security can be generically reduced to single-user security of any signature scheme with a loss in the number of users, here parties (not sessions) in the key exchange game.

Proof. Our proof of key exchange security for SIGMA-I proceeds via a sequence of code-based games [**?**]. For the first half, the proof conceptually follows the strategy put forward by Cohn-Gordon et al. [**?**].

G_1, $\boxed{\mathrm{G}_2}$

```
RunInit1($id, sk, st$):
$n_I \xleftarrow{\$} \{0,1\}^{nl}$
$x \xleftarrow{\$} \mathbb{Z}_p$
$X \leftarrow g^x$
if $(n_I, X) \in N$ then $\mathsf{bad}_C \leftarrow \mathsf{true}$ $\boxed{\text{; abort}}$
$N \leftarrow N \cup \{(n_I, X)\}$
$st'.state \leftarrow (n_I, X, x)$
$m' \leftarrow (n_I, X)$
return $(st', m')$

RunInit2($id, sk, st, peerpk, m$):
$(n_R, Y, c) \leftarrow m$
$Recv \leftarrow Recv \cup \{(n_R, Y)\}$
$(n_I, X, x) \leftarrow st.state$
...

RunResp1($id, sk, st, peerpk, m$):
$(n_I, X) \leftarrow m$
$n_R \xleftarrow{\$} \{0,1\}^{nl}$
$y \xleftarrow{\$} \mathbb{Z}_p$
$Y \leftarrow g^x$
if $(n_R, Y) \in Recv$ then $\mathsf{bad}_O \leftarrow \mathsf{true}$ $\boxed{\text{; abort}}$
if $(n_R, Y) \in N$ then $\mathsf{bad}_C \leftarrow \mathsf{true}$$\boxed{\text{; abort}}$
$N \leftarrow N \cup \{(n_R, Y)\}$
$st'.sid \leftarrow (n_I, n_R, X, Y)$
...

RO($m$):
if $H[m] = \bot$ then $H[m] \xleftarrow{\$} \{0,1\}^{kl}$
return $H[m]$
```

Figure 9: Games G_1 (changes highlighted in gray) and G_2 (changes highlighted in $\boxed{\text{frames}}$) of the SIGMA-I proof; with the explicit (lazy-sampled) random oracle RO.

Game 0. The initial game, G_0, is the key exchange security game played by $\mathcal{A}$ (cf. Figure 1), using the KGen, $\mathsf{Activate}$, and Run routines of SIGMA-I defined in Figure 8. Therefore,

$$\Pr[\mathrm{G}_0 \Rightarrow 1] = \Pr[\mathrm{G}^{\mathsf{KE\text{-}SEC}}_{\mathsf{KE},\mathcal{A}} \Rightarrow 1].$$

Game 1. Between G_0 and G_1 (Figure 9), we make internal changes to the record-keeping of the game, namely we track the nonces and group elements chosen and received by honest sessions. Whenever two honest sessions pick the same nonce or group element, we set a flag bad_C. Whenever an honest responder session picks a nonce and group element that has already been received by an initiator session, we set a flag bad_O. This change is unobservable by the adversary, hence

$$\Pr[\mathrm{G}_0 \Rightarrow 1] = \Pr[\mathrm{G}_1 \Rightarrow 1].$$

Game 2. In Game G_2 (Figure 9), we abort whenever nonces and group elements collide among honest sessions (i.e., the bad_C flag is set), or whenever an honest responder session chooses a nonce and group element already submitted by the adversary to an initiator (i.e., the bad_O flag is set). By the identical-until-bad lemma [**?**],

$$\Pr[\mathrm{G}_1 \Rightarrow 1] - \Pr[\mathrm{G}_2 \Rightarrow 1] \leq \Pr[\mathsf{bad}_C \text{ or } \mathsf{bad}_O \leftarrow \mathsf{true} \text{ in } \mathrm{G}_1].$$

In all of the calls to $\mathsf{RunInit1}$ and $\mathsf{RunResp1}$, up to q_S pairs of nonces and group elements are chosen uniformly at random. By the birthday bound, the probability of a collision between two of these pairs setting the bad_C flag is at most $\frac{q_\mathrm{S}^2}{2^{nl+1} \cdot p}$ (where nl is the nonce length and p the order of the Diffie–Hellman group). There are at most q_S pairs received by initiator sessions, so the probability that a responder session randomly chooses one of these pairs is at most $\frac{q_\mathrm{S}}{2^{nl} \cdot p}$; then by the union bound we have that $\Pr[\mathsf{bad}_O \leftarrow \mathsf{true} \text{ in } \mathrm{G}_1] \leq \frac{q_\mathrm{S}^2}{2^{nl} \cdot p}$. Since each of $\mathsf{RunInit1}$ and $\mathsf{RunResp1}$ is called at most once per SEND query, if an adversary makes q_S queries to its SEND oracle, then

$$\Pr[\mathrm{G}_1 \Rightarrow 1] - \Pr[\mathrm{G}_2 \Rightarrow 1] \leq \frac{3q_\mathrm{S}^2}{2^{nl+1} \cdot p}.$$

In all subsequent games, we are now sure that each honest session chooses a unique nonce and group element. Since the session identifier $sid = (n_I, n_R, X, Y)$ contains exactly one initiator and one responder nonce, this furthermore implies that when two honest sessions are partnered, they must have different roles.

Game 3. In Game G_3 (Figure 10), we remove the now superfluous collision flags bad_C and bad_O and add additional bookkeeping. All honest initiator sessions now log their outgoing messages in an internal table Sent. Honest responder sessions use this table to check if the message they received was sent by an honest initiator session. If so, they log their keys k_t, k_e, and k_s in a second internal table, S, indexed by their session identifier. These changes are unobservable by the adversary, so

$$\Pr[\mathrm{G}_2 \Rightarrow 1] = \Pr[\mathrm{G}_3 \Rightarrow 1].$$

Game 4. In Game G_4 (Figure 10), we require that initiator sessions whose key material has already been computed by an honest partner session simply copy their partners' key material. When an honest initiator session π_u^i with nonce n and group element X receives a message $m \leftarrow (n_R, Y, c)$, it sets its session identifier $sid \leftarrow (n_I, n_R, X, Y)$. It then checks if $\mathsf{S}[sid] \neq \perp$ (which is only the case if π_u^i has an honest partner), and if so uses the stored key material $k_s, k_t, k_e \leftarrow \mathsf{S}[st'.sid]$.

Recall that both partnered sessions agree on the DH shares X and Y as components of sid. They hence also agree on the shared DH secret $Z = g^{xy}$ and thus on the master key derived as $\mathsf{RO}(n_I \| n_R \| X \| Y \| Z)$ as well as the derived key k_s, k_t, and k_e. For the adversary $\mathcal{A}$ it is hence unobservable if initiators with honest partner actually compute their keys themselves or copy their partners' key material in Game G_4, so

$$\Pr[\mathrm{G}_3 \Rightarrow 1] = \Pr[\mathrm{G}_4 \Rightarrow 1].$$

Game 5. In Game G_5 (Figure 11), all honest sessions sample their master keys uniformly at random (as long as the random oracle has not been been queried on the corresponding input) and program the random oracle to that value (through RO's internal table $H[n_I \| n_R \| X \| Y \| Y^x] \leftarrow mk$). This is equivalent to RO performing the same checks and uniform sampling, and hence undetectable for $\mathcal{A}$:

$$\Pr[\mathrm{G}_4 \Rightarrow 1] = \Pr[\mathrm{G}_5 \Rightarrow 1].$$

Game 6. In Game G_6 (Figure 11), responder sessions whose first message came from an honest initiator stop programming the random oracle on their uniformly chosen master key mk. This is undetectable for adversary $\mathcal{A}$ unless it makes a query $\mathsf{RO}(n_I \| n_R \| X \| Y \| Z)$, where $sid = (n_I, n_R, X, Y)$ is the session identifier shared by two honest partnered sessions, and Z is the Diffie–Hellman secret corresponding to the pair (X, Y). We call this event F, and bound the probability of F by giving a reduction $\mathcal{B}_1$ (specified in Figure 12) to the strong Diffie–Hellman assumption in the DH group $\mathbb{G}$. The reduction makes at most as many queries to its stDH oracle as $\mathcal{A}$ makes to its RO oracle, as follows.

Given its strong DH challenge $(A = g^a, B = g^b)$ and having access to an oracle $\mathsf{stDH}_a(U, V)$ which outputs 1 if $U^a = V$ and 0 otherwise, $\mathcal{B}_1$ simulates G_6 for an adversary $\mathcal{A}$ as follows. In all honest initiator sessions, $\mathcal{B}_1$ embeds its challenge into the sent DH value as $X \leftarrow A \cdot g^r$, where $r \in \mathbb{Z}_p$ is sampled uniformly at random for each session. Furthermore, in all responder sessions receiving their first message from an honest initiator, $\mathcal{B}_1$ embeds its challenge as $Y \leftarrow B \cdot g^{r'}$, where $r' \in \mathbb{Z}_p$ is sampled uniformly at random for each session.

G_3, $\boxed{G_4}$

```
RunInit1($id, sk, st$):
$n_I \xleftarrow{\$} \{0,1\}^{nl}$
$x \xleftarrow{\$} \mathbb{Z}_p$
$X \leftarrow g^x$
if $(n_I, X) \in N$ then abort
$N \leftarrow N \cup \{(n_I, X)\}$
$st'.state \leftarrow (n_I, X, x)$
$m' \leftarrow (n_I, X)$
$\mathsf{Sent} \leftarrow \mathsf{Sent} \cup m'$
return $(st', m')$

RunInit2($id, sk, st, peerpk, m$):
$(n_R, Y, c) \leftarrow m$
$Recv \leftarrow Recv \cup \{(n_R, Y)\}$
$(n_I, X, x) \leftarrow st.state$
$st'.sid \leftarrow (n_I, n_R, X, Y)$
if $\mathsf{S}[st'.sid] \neq \bot$ then
    $mk \leftarrow \mathsf{RO}(n_I \| n_R \| X \| Y \| Y^x)$
    $k_s \leftarrow \mathsf{PRF}(mk, 0)$
    $k_t \leftarrow \mathsf{PRF}(mk, 1)$
    $k_e \leftarrow \mathsf{PRF}(mk, 2)$
    $\boxed{k_s, k_t, k_e \leftarrow \mathsf{S}[st'.sid]}$
else
    $mk \leftarrow \mathsf{RO}(n_I \| n_R \| X \| Y \| Y^x)$
    $k_s \leftarrow \mathsf{PRF}(mk, 0)$
    $k_t \leftarrow \mathsf{PRF}(mk, 1)$
    $k_e \leftarrow \mathsf{PRF}(mk, 2)$
$(peerid, \sigma, \tau) \leftarrow \mathsf{Dec}(k_e, c)$
$st'.peerid \leftarrow peerid$
$\ldots$

RunResp1($id, sk, st, peerpk, m$):
$(n_I, X) \leftarrow m$
$n_R \xleftarrow{\$} \{0,1\}^{nl}$
$y \xleftarrow{\$} \mathbb{Z}_p$
$Y \leftarrow g^x$
if $(n_R, Y) \in Recv$ then abort
if $(n_R, Y) \in N$ then abort
$N \leftarrow N \cup \{(n_R, Y)\}$
$st'.sid \leftarrow (n_I, n_R, X, Y)$
$\sigma \leftarrow \mathsf{S.Sign}(sk, \mathsf{L}_{rs} \| n_I \| n_R \| X \| Y)$
$mk \leftarrow \mathsf{RO}(n_I \| n_R \| X \| Y \| X^y)$
$k_s \leftarrow \mathsf{PRF}(mk, 0)$
$k_t \leftarrow \mathsf{PRF}(mk, 1)$
$k_e \leftarrow \mathsf{PRF}(mk, 2)$
if $m \in \mathsf{Sent}$ then
    $\mathsf{S}[st'.sid] \leftarrow (k_s, k_t, k_e)$
$\tau \leftarrow \mathsf{M.Tag}(k_t, \mathsf{L}_{rm} \| n_I \| n_R \| id)$
$st'.state \leftarrow (n_I, n_R, X, Y, k_s, k_t)$
$m' \leftarrow (n_R, Y, \mathsf{Enc}(k_e, (id, \sigma, \tau)))$
return $(st', m')$
```

Figure 10: Games G_3 (changes highlighted in gray) and G_4 (changes highlighted in $\boxed{\text{frames}}$) of the SIGMA-I proof.

Let us first observe that if event F occurs, then the value Z in the random oracle query $\mathsf{RO}(n_I \| n_R \| X \| Y \| Z)$ will equal $g^{(a+r)(b+r')}$ for some $r, r' \in \mathbb{Z}_p$ chosen by $\mathcal{B}_1$, and consequently

$$Z \cdot Y^{-r} = g^{(a+r)(b+r')-(b+r')\cdot r} = g^{a(b+r')} = Y^a.$$

This equality can be tested for by $\mathcal{B}_1$ by calling its stDH_a oracle on the pair $(Y, Z \cdot Y^{-r})$. We let $\mathcal{B}_1$ do so whenever $\mathcal{A}$ queries RO on some value $(n_I \| n_R \| X \| Y \| Z)$ where $(n_I, X = A \cdot g^r)$ was output by an honest initiator session and $(n_R, Y = g^{(b+r')})$ was output by a responder session with an honest initiator; the responder stores (n_I, n_R, X, Y) in a look-up table Q so this can be checked efficiently. If $\mathsf{stDH}_a(Y, Z \cdot Y^{-r}) = 1$ on such occasion, i.e., event F occurs, $\mathcal{B}_1$ stops with output $Z \cdot Y^{-r} \cdot A^{-r'} = g^{(a+r)(b+r')} \cdot g^{-(b+r')\cdot r} \cdot g^{-ar'} = g^{ab}$ and wins. Therefore,

$$\Pr[F] \leq \mathsf{Adv}^{\mathsf{stDH}}_{\mathbb{G},\mathcal{B}_1}.$$

One subtlety in this step is ensuring that $\mathcal{B}_1$ can correctly simulate answers to RevSessionKey queries to any initiator or responder session. We do so by accordingly programming the random oracle on the sampled master key, where needed. First of all observe that responder sessions without honest initiator keep picking their own Y share and compute mk regularly. Initiator and responder sessions with honest partner have the challenge embedded and sample an independent master key which is not programmed to the random oracle. However, $\mathcal{B}_1$ stops and wins (as described above) if $\mathcal{A}$ ever queries the random oracle on the correct DH secret; i.e., $\mathcal{A}$ never sees the (inconsistent) random oracle output for these master keys. The interesting case is when an initiator session (which always embeds the challenge in its DH share as $X = A \cdot g^r$) obtains a message (n_R, Y, c) *not* originating from an honest responder: Here, Y may well have been picked by the adversary who could furthermore have corrupted the initiator's peer and hence make the initiator accept—with a master key it cannot compute itself.

We therefore let $\mathcal{B}_1$ attempt to copy the adversary's master key, if it has been computed. The RO oracle logs all queries it receives by their putative session id (n_I, n_R, X, Y) in a look-up table H', so $\mathcal{B}_1$ can efficiently access all Z such that (n_I, n_R, X, Y, Z) has been queried to RO. Since the DH secret corresponding to the pair (X, Y) equals Y^{a+r}, if Z is this DH secret, then

$$Z \cdot Y^{-r} = Y^{(a+r)-r} = Y^a.$$

The reduction can check this equality using its stDH_a oracle and in that case use the response to $\mathsf{RO}(n_I, n_R, X, Y, Z)$ as mk. Otherwise, $\mathcal{B}_1$ samples mk at random and stores it in the table Q (Line 48 of Figure 12), indicating it should be programmed in the random oracle later if queried on a matching Z value (Line 75). This ensures all responses to RevSessionKey queries are consistent with $\mathcal{A}$'s queries to the random oracle RO.

Observe that, in all this, $\mathcal{B}_1$ calls its stDH oracle at most once for each entry $H[n_I \| n_R \| X \| Y \| Z] = mk$ in the RO table H. In RO, stDH is called (once) only for entries that were not present when $\mathsf{Q}[(n_I, n_R, X, Y)]$ was set, then H' is set. In $\mathsf{RunInit2}$ and $\mathsf{RunResp1}$, stDH is called only for matching H' entries established prior to setting Q. Therefore, if stDH is called in RO for an entry, it was not called in either $\mathsf{RunInit2}$ or $\mathsf{RunResp1}$. If stDH is called on an entry in $\mathsf{RunResp1}$, then the responder session is partnered, so its partner will copy its keys in $\mathsf{RunInit2}$ and not call stDH. Furthermore, due to uniqueness of nonces and DH shares (by Game G_2), no $\mathsf{RunInit2}$ or $\mathsf{RunResp1}$ call makes stDH be invoked twice for the same H' entry.

Since the total time to iterate through the for loops over all Run and RO queries is at most $O(q_{\mathrm{RO}})$, the running time of $\mathcal{B}_1$ is roughly that of $\mathcal{A}$, plus the time needed to compute the arguments of the stDH queries. Each of these arguments requires one group operation and one exponentiation.

G_5, $\boxed{G_6}$

```
RunInit2($id, sk, st, peerpk, m$):
$(n_R, Y, c) \leftarrow m$
$Recv \leftarrow Recv \cup \{(n_R, Y)\}$
$(n_I, X, x) \leftarrow st.state$
$st'.sid \leftarrow (n_I, n_R, X, Y)$
if $\mathsf{S}[st'.sid] \neq \bot$ then
    $k_s, k_t, k_e \leftarrow \mathsf{S}[st'.sid]$
else
    $mk \xleftarrow{\$} \{0,1\}^{kl}$
    if $H[n_I \| n_R \| X \| Y \| Y^x] \neq \bot$
        $mk \leftarrow H[n_I \| n_R \| X \| Y \| Y^x]$
    $H[n_I \| n_R \| X \| Y \| Y^x] \leftarrow mk$
    $k_s \leftarrow \mathsf{PRF}(mk, 0)$
    $k_t \leftarrow \mathsf{PRF}(mk, 1)$
    $k_e \leftarrow \mathsf{PRF}(mk, 2)$
$(peerid, \sigma, \tau) \leftarrow \mathsf{Dec}(k_e, c)$
$st'.peerid \leftarrow peerid$
if $\mathsf{S.Vrfy}(peerpk[peerid], \mathsf{L}_{rs} \| n_I \| n_R \| X \| Y, \sigma)$
and $\mathsf{M.Vrfy}(k_t, \mathsf{L}_{rm} \| n_I \| n_R \| peerid)$ then
    $st'.status \leftarrow \mathsf{accepted}$
    $st'.skey \leftarrow k_s$
    $\sigma' \leftarrow \mathsf{S.Sign}(sk, \mathsf{L}_{is} \| n_I \| n_R \| X \| Y)$
    $\tau' \leftarrow \mathsf{M.Tag}(k_t, \mathsf{L}_{im} \| n_I \| n_R \| id)$
    $m' \leftarrow \mathsf{Enc}(k_e, (id, \sigma', \tau'))$
else
    $m' \leftarrow \bot$ ; $st'.status \leftarrow \mathsf{rejected}$
return $(st', m')$
```

```
RunResp1($id, sk, st, peerpk, m$):
$(n_I, X) \leftarrow m$
$n_R \xleftarrow{\$} \{0,1\}^{nl}$
$y \xleftarrow{\$} \mathbb{Z}_p$
$Y \leftarrow g^x$
if $(n_R, Y) \in Recv$ then abort
if $(n_R, Y) \in N$ then abort
$N \leftarrow N \cup \{(n_R, Y)\}$
$st'.sid \leftarrow (n_I, n_R, X, Y)$
$\sigma \leftarrow \mathsf{S.Sign}(sk, \mathsf{L}_{rs} \| n_I \| n_R \| X \| Y)$
$mk \xleftarrow{\$} \{0,1\}^{kl}$
$\boxed{\text{if } m \notin \mathsf{Sent} \text{ then}}$
    if $H[n_I \| n_R \| X \| Y \| X^y] \neq \bot$
        $mk \leftarrow H[n_I \| n_R \| X \| Y \| X^y]$
    $H[n_I \| n_R \| X \| Y \| X^y] \leftarrow mk$
$k_s \leftarrow \mathsf{PRF}(mk, 0)$
$k_t \leftarrow \mathsf{PRF}(mk, 1)$
$k_e \leftarrow \mathsf{PRF}(mk, 2)$
if $m \in \mathsf{Sent}$ then
    $\mathsf{S}[st'.sid] \leftarrow (k_s, k_t, k_e)$
$\tau \leftarrow \mathsf{M.Tag}(k_t, \mathsf{L}_{rm} \| n_I \| n_R \| id)$
$st'.state \leftarrow (n_I, n_R, X, Y, k_s, k_t)$
$m' \leftarrow (n_R, Y, id, \sigma, \tau)$
return $(st', m')$
```

Figure 11: Games G_5 (changes highlighted in gray) and G_6 (changes highlighted in $\boxed{\text{frames}}$) of the SIGMA-I proof.

```
$\mathcal{B}_1(A, B)^{\mathsf{stDH}_a(\cdot,\cdot)}$

$\mathsf{RunInit1}(id, sk, st)$:
$n_I \xleftarrow{\$} \{0,1\}^{nl}$
$r \xleftarrow{\$} \mathbb{Z}_p$
$X \leftarrow A \cdot g^r$
if $(n_I, X) \in N$ then abort
$N \leftarrow N \cup \{(n_I, X)\}$
$st'.state \leftarrow (n_I, X, r)$
$m' \leftarrow (n_I, X)$
$\mathsf{Sent}[m'] \leftarrow x$
return $(st', m')$

$\mathsf{RunInit2}(id, sk, st, peerpk, m)$:
$(n_R, Y, c) \leftarrow m$
$Recv \leftarrow Recv \cup \{(n_R, Y)\}$
$(n_I, X, r) \leftarrow st.state$
$st'.sid \leftarrow (n_I, n_R, X, Y)$
if $\mathsf{S}[st'.sid] \neq \perp$ then
    $k_s, k_t, k_e \leftarrow \mathsf{S}[st'.sid]$
else
    $mk \xleftarrow{\$} \{0,1\}^{kl}$
    for each $Z \in H'[n_I \| n_R \| X \| Y]$
        if $\mathsf{stDH}_a(Y, Z \cdot Y^{-r}) = 1$ then
            $mk \leftarrow H[n_I \| n_R \| X \| Y \| Z]$
    $\mathsf{Q}[st'.sid] \leftarrow (r, \perp, mk)$
    $k_s \leftarrow \mathsf{PRF}(mk, 0)$
    $k_t \leftarrow \mathsf{PRF}(mk, 1)$
    $k_e \leftarrow \mathsf{PRF}(mk, 2)$
$(peerid, \sigma, \tau) \leftarrow \mathsf{Dec}(k_e, c)$
$st'.peerid \leftarrow peerid$
if $\mathsf{S.Vrfy}(\mathsf{L}_{rs} \| n_I \| n_R \| X \| Y, \sigma, pk_{peerid})$
and $\mathsf{M.Vrfy}(k_t, \mathsf{L}_{rm} \| n_I \| n_R \| peerid)$ then
    $st'.status \leftarrow$ accepted
    $st'.skey \leftarrow k_s$
    $\sigma' \leftarrow \mathsf{S.Sign}(sk, \mathsf{L}_{is} \| n_I \| n_R \| R \| W)$
    $\tau' \leftarrow \mathsf{M.Tag}(k_t, \mathsf{L}_{im} \| n_I \| n_R \| id)$
    $m' \leftarrow \mathsf{Enc}(k_e, (id, \sigma', \tau'))$
else
    $m' \leftarrow \perp$
    $st'.status \leftarrow$ rejected
return $(st', m')$
```

```
$\mathsf{RunResp1}(id, sk, st, peerpk, m)$:
$(n_I, X) \leftarrow m$
$n_R \xleftarrow{\$} \{0,1\}^{nl}$
$r' \xleftarrow{\$} \mathbb{Z}_p$
$mk \xleftarrow{\$} \{0,1\}^{kl}$
if $m \in \mathsf{Sent}$ then
    $r \leftarrow \mathsf{Sent}[m]$
    $Y \leftarrow B \cdot g^{r'}$
    $st'.sid \leftarrow (n_I, n_R, X, Y)$
    for each $Z \in H'[n_I \| n_R \| X \| Y]$
        if $\mathsf{stDH}_a(Y, Z \cdot Y^{-r}) = 1$ then
            $\textsc{Finalize}(Z \cdot Y^{-r} \cdot A^{-r'})$
    $\mathsf{Q}[st'.sid] \leftarrow (r, r', mk)$
else
    $Y \leftarrow g^{r'}$
    $st'.sid \leftarrow (n_I, n_R, X, Y)$
    if $H[n_I \| n_R \| X \| Y \| X^y] \neq \perp$
        $mk \leftarrow H[n_I \| n_R \| X \| Y \| X^{r'}]$
    $H[n_I \| n_R \| X \| Y \| X^y] \leftarrow mk$
if $(n_R, Y) \in Recv$ then abort
if $(n_R, Y) \in N$ then abort
$N \leftarrow N \cup \{(n_R, Y)\}$
$\sigma \leftarrow \mathsf{S.Sign}(sk, \mathsf{L}_{rs} \| n_I \| n_R \| X \| Y)$
$k_s \leftarrow \mathsf{PRF}(mk, 0)$
$k_t \leftarrow \mathsf{PRF}(mk, 1)$
$k_e \leftarrow \mathsf{PRF}(mk, 2)$
if $m \in \mathsf{Sent}$ then
    $\mathsf{S}[st'.sid] \leftarrow (k_s, k_t, k_e)$
$\tau \leftarrow \mathsf{M.Tag}(k_t, \mathsf{L}_{rm} \| n_I \| n_R \| id)$
$st'.state \leftarrow (n_I, n_R, X, Y, k_s, k_t, k_e)$
$m' \leftarrow (n_R, Y, \mathsf{Enc}(k_e, (id, \sigma, \tau)))$
return $(st', m')$

$\mathsf{RO}(m)$:
if $H[m] = \perp$ then
    $H[m] \xleftarrow{\$} \{0,1\}^{kl}$
    parse $n_I \| n_R \| X \| Y \| Z \leftarrow m$
    $H'[n_I \| n_R \| X \| Y] \leftarrow H'[n_I \| n_R \| X \| Y] \cup \{Z\}$
    if $\mathsf{Q}[(n_I, n_R, X, Y)] \neq \perp$ then
        $(r, r', mk) \leftarrow \mathsf{Q}[n_I, n_R, X, Y]$
        if $\mathsf{stDH}_a(Y, Z \cdot Y^{-r}) = 1$ then
            if $r' = \perp$ then $H[m] \leftarrow mk$
            else $\textsc{Finalize}(Z \cdot Y^{-r} \cdot A^{-r'})$
return $H[m]$
```

Figure 12: Reduction $\mathcal{B}_1$ to the strong Diffie–Hellman assumption of the SIGMA-I proof. Sections highlighted in gray have been significantly altered compared to Game G_6.

$\underline{\mathrm{G}_7}$

```
RunResp1$(id, sk, st, peerpk, m)$:
$(n_I, X) \leftarrow m$
$n_R \xleftarrow{\$} \{0,1\}^{nl}$
$y \xleftarrow{\$} \mathbb{Z}_p$
$Y \leftarrow g^y$
if $(n_R, Y) \in Recv$ then abort
if $(n_R, Y) \in N$ then abort
$N \leftarrow N \cup \{(n_R, Y)\}$
$st'.sid \leftarrow (n_I, n_R, X, Y)$
$\sigma \leftarrow \mathsf{S.Sign}(sk, \mathsf{L}_{rs} \| n_I \| n_R \| X \| Y)$
$mk \xleftarrow{\$} \{0,1\}^{kl}$
if $m \notin \mathsf{Sent}$ then
    if $H[Y^x \| n_I \| n_R \| X \| Y] \neq \bot$
        $mk \leftarrow H[Y^x \| n_I \| n_R \| X \| Y]$
$k_s \leftarrow \mathsf{PRF}(mk, 0)$
$k_t \leftarrow \mathsf{PRF}(mk, 1)$
$k_e \leftarrow \mathsf{PRF}(mk, 2)$
if $m \in \mathsf{Sent}$
    $k_s \xleftarrow{\$} \{0,1\}^{kl}$
    $k_t \xleftarrow{\$} \{0,1\}^{kl}$
    $k_e \xleftarrow{\$} \{0,1\}^{kl}$
    $\mathsf{S}[st'.sid] \leftarrow (k_s, k_t, k_e)$
$\tau \leftarrow \mathsf{M.Tag}(k_t, \mathsf{L}_{rm} \| n_I \| n_R \| id)$
$st'.state \leftarrow (n_I, n_R, X, Y, k_s, k_t)$
$m' \leftarrow (n_R, Y, \mathsf{Enc}(k_e, (id, \sigma, \tau)))$
return $(st', m')$
```

$\underline{\mathcal{B}_2^{\textsc{Fn}(\cdot,\cdot)}}$

```
RunResp1$(id, sk, st, peerpk, m)$:
$(n_I, X) \leftarrow m$
$n_R \xleftarrow{\$} \{0,1\}^{nl}$
$y \xleftarrow{\$} \mathbb{Z}_p$
$Y \leftarrow g^y$
if $(n_R, Y) \in Recv$ then abort
if $(n_R, Y) \in N$ then abort
$N \leftarrow N \cup \{(n_R, Y)\}$
$st'.sid \leftarrow (n_I, n_R, X, Y)$
$\sigma \leftarrow \mathsf{S.Sign}(sk, \mathsf{L}_{rs} \| n_I \| n_R \| X \| Y)$
$mk \xleftarrow{\$} \{0,1\}^{kl}$
if $m \notin \mathsf{Sent}$ then
    if $H[n_I \| n_R \| X \| Y \| X^y] \neq \bot$
        $mk \leftarrow H[n_I \| n_R \| X \| Y \| X^y]$
$k_s \leftarrow \mathsf{PRF}(mk, 0)$
$k_t \leftarrow \mathsf{PRF}(mk, 1)$
$k_e \leftarrow \mathsf{PRF}(mk, 2)$.
if $m \in \mathsf{Sent}$
    $\textsc{New}();\ i{+}{+}$
    $k_s \leftarrow \textsc{Fn}(i, 0)$
    $k_t \leftarrow \textsc{Fn}(i, 1)$
    $k_e \leftarrow \textsc{Fn}(i, 2)$
    $\mathsf{S}[st'.sid] \leftarrow (k_s, k_t, k_e)$
$\tau \leftarrow \mathsf{M.Tag}(k_t, \mathsf{L}_{rm} \| n_I \| n_R \| id)$
$st'.state \leftarrow (n_I, n_R, X, Y, k_s, k_t)$
$m' \leftarrow (n_R, Y, \mathsf{Enc}(k_e, (id, \sigma, \tau)))$
return $(st', m')$
```

Figure 13: Game G_7 and reduction $\mathcal{B}_2$ to PRF security of the SIGMA-I proof. Changes from G_6 resp. compared to G_7 highlighted in gray.

(All other operations performed by $\mathcal{B}_1$ add only a small constant amount of time per SEND query, which is dominated by the runtime of $\mathcal{A}$.) The exponentiation can be computed using $2\log_2 p$ group operations using the square-and-multiply (or double-and-add) algorithm, so $t_{\mathcal{B}_1} \approx t + 2q_{\mathrm{RO}} \log_2 p$. The runtime t of $\mathcal{A}$ already includes the computation of $2q_{\mathrm{S}} \log_2 p$ group operations, so this is a significant but not prohibitive increase in runtime.

Having $\mathcal{B}_1$ perfectly simulate Game G_5 for $\mathcal{A}$ up to the point when F happens, and G_6 and G_5 differing only when F happens, we have

$$\Pr[\mathrm{G}_5 \Rightarrow 1] = \Pr[\mathrm{G}_6 \Rightarrow 1] + \Pr[F] \leq \Pr[\mathrm{G}_6 \Rightarrow 1] + \mathsf{Adv}^{\mathsf{stDH}}_{\mathbb{G},}(t_{\mathcal{B}_1}, q_{\mathrm{RO}}),$$

and $t_{\mathcal{B}_1} \approx t + 2q_{\mathrm{RO}} \log_2 p$.

Game 7. In Game G_7 (Figure 13), responder oracles responding to honest messages samples session, MAC, and encryption keys k_s, k_t, and k_e randomly instead of computing them through a PRF. (Initiator oracles partnered with an honest responder will continue to copy those, now randomly sampled keys.)

Since the PRF key mk in this case is sampled independently of the random oracle and the rest of the game, this reduces straightforwardly to the multi-user security of the PRF via the reduction $\mathcal{B}_2$ we give in Figure 13. The adversary $\mathcal{B}_2$ makes one New and two FUNC queries for each $\mathsf{RunResp1}$ query, or three FUNC queries in SIGMA-I. Notably, it makes at most three FUNC queries per user, and no Corrupt queries because mk is never revealed to the adversary. Outside of the oracle calls, its running time exactly equals that of $\mathcal{A}$ in Game G$_6$, as their pseudocode is identical, so $t_{\mathcal{B}_2} \approx t$. Using its Fn oracle of the PRF game, $\mathcal{B}_2$ perfectly simulates G$_6$ if the oracle gives real-PRF answers and G$_7$ if it returns uniformly random values. Therefore,

$$\Pr[\mathrm{G}_6 \Rightarrow 1] \leq \Pr[\mathrm{G}_7 \Rightarrow 1] + \mathsf{Adv}^{\text{mu-PRF}}_{\text{PRF},}(t_{\mathcal{B}_2}, q_{\mathrm{S}}, 3q_{\mathrm{S}}, 3, 0).$$

Observe that from now on, session and MAC keys of responder oracles that received honest initiator's messages are chosen independently at random, and that initiator oracles with matching sid will copy those keys. Notably, this is the case even for sessions whose (own or peer's) long-term secret have been revealed to the adversary. We will use these properties in the following to argue authentication of sessions as well as forward security of the session keys.

Our final game hops are concerned with the explicit authentication performed through signatures and MACs in the SIGMA-I protocol, and as such extend those proof steps for implicit authentication of the main protocols in [**?**].

Game 8. In Game G$_8$ (Figure 14), we log all messages for which signatures are generated by an honest session, and set a bad flag bad_S if the adversary submits a valid signature under an uncorrupted signing key for a message which was not produced by an honest session. This internal bookkeeping does not affect the adversary's advantage, so

$$\Pr[\mathrm{G}_7 \Rightarrow 1] = \Pr[\mathrm{G}_8 \Rightarrow 1].$$

Game 9. In Game G$_9$ (Figure 14), we abort if the bad_S flag is set. By the identical-until-bad lemma, the difference in advantage between G$_8$ and G$_9$ is bounded by the probability that this event occurs, which we reduce via an algorithm $\mathcal{B}_3$ to the multi-user security of the digital signature scheme S.

In the reduction, $\mathcal{B}_3$ obtains all long-term public keys from the multi-user signature game and uses its signing oracles for any honest signature to be produced. It therefore makes q_{N} queries to New and one Sign query for each call to $\mathsf{RunResp1}$ or $\mathsf{RunInit2}$, for at most q_{S} such queries. It relays RevLongTermKey queries as corruptions in its multi-user game, making q_{RL} corruption queries in total. When bad_S is triggered, $\mathcal{B}_3$ submits the triggering message and signature under the targeted (uncorrupted) public key as its forgery. As the triggering message was not signed before under the corresponding secret key (and hence not queried to the signing oracle by $\mathcal{B}_3$), the forgery is valid and $\mathcal{B}_3$ wins if bad_S is set. It follows that

$$\Pr[\mathrm{G}_8 \Rightarrow 1] \leq \Pr[\mathrm{G}_9 \Rightarrow 1] + \mathsf{Adv}^{\text{mu-EUF-CMA}}_{\mathsf{S},\mathcal{B}_3}(t_{\mathcal{B}_3}, q_{\mathrm{N}}, q_{\mathrm{S}}, q_{\mathrm{S}}, q_{\mathrm{RL}}).$$

Except for the replacement of key generation, signatures, corruptions with oracle queries, the pseudocode of $\mathcal{B}_3$ is identical to that of $\mathcal{A}$ in game G$_8$, so $t_{\mathcal{B}_3} \approx t$.

Game 10. In Game G$_{10}$ (Figure 14), we remove the now redundant bad_S flag again, and log all MAC tags generated by honest sessions with honest partners in a list Q_M (using, as before, the table S to determine whether a session has an honest partner). We set a flag bad_M if a session with

G_8, $\boxed{\mathrm{G}_9}$

```
RunInit2($id, sk, st, peerpk, m$):
...
if S.Vrfy($peerpk[peerid], \mathsf{L}_{rs} \| n_I \| n_R \| X \| Y, \sigma$)
and M.Vrfy($k_t, \mathsf{L}_{rm} \| n_I \| n_R \| peerid, \tau$) then
    if revltk$_{peerid} = \infty$ and
        $(peerid, \mathsf{L}_{rs} \| n_I \| n_R \| X \| Y) \notin \mathsf{Q}_S$ then
        bad$_S$ ← true $\boxed{\text{; abort}}$
    $st'.status$ ← accepted
    $st'.skey \leftarrow k_s$
    $\sigma' \leftarrow$ S.Sign($sk, \mathsf{L}_{is} \| n_I \| n_R \| X \| Y$)
    $\mathsf{Q}_S \leftarrow \mathsf{Q}_S \cup \{(id, \mathsf{L}_{is} \| n_I \| n_R \| X \| Y)\}$
    $\tau' \leftarrow$ M.Tag($k_t, \mathsf{L}_{im} \| n_I \| n_R \| id$)
...

RunResp1($id, sk, st, peerpk, m$):
...
$\sigma \leftarrow$ S.Sign($sk, \mathsf{L}_{rs} \| n_I \| n_R \| X \| Y$)
$\mathsf{Q}_S \leftarrow \mathsf{Q}_S \cup \{(id, \mathsf{L}_{rs} \| n_I \| n_R \| X \| Y)\}$
...
$\tau \leftarrow$ M.Tag($k_t, \mathsf{L}_{rm} \| n_I \| n_R \| id$)
...

RunResp2($id, sk, st, peerpk, m$):
...
if S.Vrfy($peerpk[peerid], \mathsf{L}_{is} \| n_I \| n_R \| X \| Y, \sigma'$)
and M.Vrfy($k_t, \mathsf{L}_{im} \| n_I \| n_R \| peerid, \tau'$) then
    if revltk$_{peerid} = \infty$ and
        $(peerid, \mathsf{L}_{rs} \| n_I \| n_R \| X \| Y) \notin \mathsf{Q}_S$ then
        bad$_S$ ← true $\boxed{\text{; abort}}$
    $st'.status$ ← accepted
    $st'.skey \leftarrow k_s$
else $st'.status$ ← rejected
return $(st', m')$
```

G_{10}, $\boxed{\mathrm{G}_{11}}$

```
RunInit2($id, sk, st, peerpk, m$):
...
if S.Vrfy($peerpk[peerid], \mathsf{L}_{rs} \| n_I \| n_R \| X \| Y, \sigma$)
and M.Vrfy($k_t, \mathsf{L}_{rm} \| n_I \| n_R \| peerid$) then
    if revltk$_{peerid} = \infty$ and
        $(peerid, \mathsf{L}_{rs} \| n_I \| n_R \| X \| Y) \notin \mathsf{Q}_S$ then
        abort
    if S$[st'.sid] \neq \perp$ and
        $(st'.sid, \mathsf{L}_{rm} \| n_I \| n_R \| peerid) \notin \mathsf{Q}_M$ then
        bad$_M$ ← true $\boxed{\text{; abort}}$
    $st'.status$ ← accepted
    $st'.skey \leftarrow k_s$
    $\sigma' \leftarrow$ S.Sign($sk, \mathsf{L}_{is} \| n_I \| n_R \| X \| Y$)
    $\mathsf{Q}_S \leftarrow \mathsf{Q}_S \cup \{(id, \mathsf{L}_{is} \| n_I \| n_R \| X \| Y)\}$
    $\tau' \leftarrow$ M.Tag($k_t, \mathsf{L}_{im} \| n_I \| n_R \| id$)
    if S$[st'.sid] \neq \perp$ then
        $\mathsf{Q}_M \leftarrow \mathsf{Q}_M \cup \{(st'.sid, \mathsf{L}_{im} \| n_I \| n_R \| id)\}$
...

RunResp1($id, sk, st, peerpk, m$):
...
$\sigma \leftarrow$ S.Sign($sk, \mathsf{L}_{rs} \| n_I \| n_R \| X \| Y$)
$\mathsf{Q}_S \leftarrow \mathsf{Q}_S \cup \{(id, \mathsf{L}_{rs} \| n_I \| n_R \| X \| Y)\}$
...
$\tau \leftarrow$ M.Tag($k_t, \mathsf{L}_{rm} \| n_I \| n_R \| id$)
if S$[st'.sid] \neq \perp$ then
    $\mathsf{Q}_M \leftarrow \mathsf{Q}_M \cup \{(st'.sid, \mathsf{L}_{rm} \| n_I \| n_R \| id)\}$
...

RunResp2($id, sk, st, peerpk, m$):
...
if S.Vrfy($peerpk[peerid], \mathsf{L}_{is} \| n_I \| n_R \| X \| Y, \sigma'$)
and M.Vrfy($k_t, \mathsf{L}_{im} \| n_I \| n_R \| peerid, \tau'$) then
    if revltk$_{peerid} = \infty$ and
            $(peerid, \mathsf{L}_{is} \| n_I \| n_R \| X \| Y) \notin \mathsf{Q}_S$ then
        abort
    if $S[st'.sid] \neq \perp$ and
        $(st'.sid, (peerid, \mathsf{L}_{im} \| n_I \| n_R \| peerid) \notin \mathsf{Q}_M$ then
        bad$_M$ ← true $\boxed{\text{; abort}}$
    $st'.status$ ← accepted
    $st'.skey \leftarrow k_s$
else $st'.status$ ← rejected
return $(st', m')$
```

Figure 14: Games G_8, G_9, G_{10}, and G_{11} of the SIGMA-I proof. Changes in G_8 and G_{10} are highlighted in gray, changes in G_9 and G_{11} are highlighted in $\boxed{\text{frames}}$.

an honest partner receives a valid MAC tag which was not computed by any honest oracle. This bookkeeping is similar to the changes from G_7 to G_8, but noting MAC tags instead of signatures. As before, the bookkeeping itself does not affect the adversary's advantage:

$$\Pr[\mathrm{G}_9 \Rightarrow 1] = \Pr[\mathrm{G}_{10} \Rightarrow 1].$$

Game 11. In Game G_{11} (Figure 14), we abort if the bad_M flag is set to true. Again applying the identical-until-bad lemma, we need to bound the probability of bad_M being set in G_{10}, which we do via the following reduction $\mathcal{B}_4$ to the multi-user EUF-CMA security of the MAC scheme M.

The reduction $\mathcal{B}_4$ simulates G_{10} truthfully, except that for any session with honest origin partner (i.e., session with state st where $\mathsf{S}[st.sid] \neq \perp$), $\mathcal{B}_4$ does not compute k_t itself, but instead assigns an incremented user identifier i to this session's sid and computes any calls to Tag or Vrfy using its corresponding oracles for user i. There is at most one query to $\textsc{NewUser}$, and one each to Tag and Vrfy for each of $\mathcal{A}$'s queries to $\textsc{Send}$. Hence $\mathcal{B}_4$ makes at most q_{S} queries to each of these three oracles, and at most one query to Tag and Vrfy per user in the mu-EUF-CMA game. When bad_M is triggered, $\mathcal{B}_4$ submits the triggering message and MAC tag under user identifier i as its forgery. In the simulation, sessions will share a user identifier i if and only if they are partnered and would share keys in Game G_{10}. These keys are furthermore unique to one initiator and one responder session only, so consistency is maintained. Furthermore, k_t cannot be exposed (by $\textsc{RevLongTermKey}$ or $\textsc{RevSessionKey}$) to adversary $\mathcal{A}$, hence implicitly replacing it with the MAC game's oracles is sound, and $\mathcal{B}_4$ makes no $\textsc{Corrupt}$ queries. Except for oracle replacements, the pseudocode of $\mathcal{B}_4$ is identical to that of $\mathcal{A}$ in G_{10}, so $t_{\mathcal{B}_4} \approx t$.

If bad_M is triggered, then $S[st'.sid] \neq \perp$, so $st'.sid$ corresponds to some user identifier i in the multi-user EUF-CMA game. Additionally, a tag τ for message m was verified under identity i, and $(st'.sid, m)$ was not logged in Q_M. Since $\mathcal{B}_4$ logs $(st'.sid, m)$ every time it calls its Tag oracle on the pair (i, m), this call cannot have occurred. Then τ is a valid forgery on m, which $\mathcal{B}_4$ will output for user i to win the EUF-CMA game. Thus,

$$\Pr[\mathrm{G}_{10} \Rightarrow 1] \leq \Pr[\mathrm{G}_{11} \Rightarrow 1] + \mathsf{Adv}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{M},}(t_{\mathcal{B}_4}, q_{\mathrm{S}}, q_{\mathrm{S}}, 1, q_{\mathrm{S}}, 1, 0).$$

We can now consider the final advantage of an adversary playing Game G_{11}. Adversary $\mathcal{A}$ has a non-zero advantage if in the final oracle query $\textsc{Finalize}(b')$

1. Sound is false,
2. $\mathsf{ExplicitAuth}$ is false, or
3. Fresh is true and $b' = b$.[5]

Soundness. The flag Sound is set if (1) three honest sessions hold the same session identifier, or if (2) two partnered sessions hold different session keys.

For (1): No three honest sessions can share the same session identifiers, as this would require a collision in either the contained initiator or responder nonce, which is excluded by Game G_2.

For (2): The session identifier includes both nonces n and n_R and DH shares X and Y, which together determine the derived master key $mk = \mathsf{RO}(n_I \| n_R \| X \| Y \| Z)$ (where Z is the DH secret from X and Y) and thus the session key. Agreement on the session identifier hence implies deriving the same session key.

Hence, in Game G_{11}, Sound is always true.

[5] If Fresh is false, $b = b' = 0$ happens with probability $\frac{1}{2}$, so $\mathcal{A}$'s advantage is 0.

Explicit authentication. The predicate $\mathsf{ExplicitAuth}$ requires that for any session π_u^i accepting with a non-compromised peer v, there exists a partnered session π_v^j of user v with opposite role which, if it accepts, has u set as its peer.

The session π_u^i, prior to accepting, obtained a valid signature on $\pi_u^i.sid$ and a label corresponding to a role $r \neq \pi_u^i.role$. Due to Game G_9, this signature must have been issued by an honest session π_v^j (since v was not compromised at this point). All honest sessions sign their own sid and a label corresponding to their own role, so $\pi_u^i.sid = \pi_v^j.sid$ and $\pi_u^i.role = r \neq \pi_v^j.role$ are satisfied.

Furthermore, when π_v^j accepts, it must have received a valid MAC tag τ on a label identifying an opposite-role session and that session's user identity, as well as their shared nonces. Due to Game G_{11}, this MAC value must have been computed by an honest session holding the same nonces, as π_v^j has an honest partner session and therefore $\mathsf{S}[\pi_v^j.sid] \neq \bot$. Furthermore, by Game G_2, nonces do not collide and hence that session must have been π_u^i, thus computing the MAC on user identity u, which π_v^j accordingly sets as peer identity.

Therefore $\mathsf{ExplicitAuth}$ is always true in G_{11}. Note that we did not require that the long-term key of user u was uncorrupted, and we allow the adversary to continue interacting with sessions after compromise; hence covering key compromise impersonation attacks.

Guessing the challenge bit. Finally, we have to consider $\mathcal{A}$'s chance of guessing the challenge bit b, which it may only learn through TEST queries such that all tested sessions are fresh (i.e., Fresh is true).

The Fresh predicate being true ensures that all tested sessions (those in T) accepted prior to their respective partner being corrupt. Then, as $\mathsf{ExplicitAuth}$ is true, we have that for each tested session there exists an honest session with the same sid and different roles. This session, by Fresh, was not tested or revealed. Being partnered, the first message (n_I, X) between these two honest sessions was not tampered with, so in the responder session, whether it was the tested session or its partner, the master and session keys are sampled uniformly at random (due to Games G_6 and G_7). Since the initiator session holds the same sid, it copied the responder's random session key (due to Game G_4). This random session key was not revealed in either of the two sessions (by Fresh), and hence from $\mathcal{A}$'s perspective is a uniformly random and independent value. In all TEST oracle responses, k_0 and k_1 are hence identically distributed and so G_{11} is fully independent of b. It follows that the adversary $\mathcal{A}$ has no better than a $\frac{1}{2}$ probability of choosing b' equal to b, so

$$\Pr[\mathrm{G}_{11} \Rightarrow 1] = \frac{1}{2},$$

which concludes the proof. □

7 The TLS 1.3 Handshake Protocol

The Transport Layer Security (TLS) protocol in version 1.3 [?] bases its key exchange design (the so-called handshake protocol) on a variant of SIGMA-I. Following the core SIGMA design, the TLS 1.3 main handshake is an ephemeral Diffie–Hellman key exchange, authenticated through a combination of signing and MAC-ing the (full, hashed) communication transcript.[6] Additionally, and similar to SIGMA-I, beyond establishing the main (application traffic) session key, handshake traffic keys are derived and used to encrypt part of the handshake.

[6] TLS 1.3 also specifies an abbreviated resumption-style handshake based on pre-shared keys; we focus on the main DH-based handshake in this work.

Beyond additional protocol features like negotiating the cryptographic algorithms to be used, communicating further information in extensions, etc.—which we do not capture here—, TLS 1.3 however deviates in two core cryptographic aspects from the more simplistic and abstract SIGMA(-I) design: it hashes the communication transcript when deriving keys and computing signatures and MACs, and it uses a significantly more complicated key schedule. In this section we revisit the TLS 1.3 handshake and discuss the careful technical changes and additional assumptions needed to translate our tight security results for SIGMA-I to TLS 1.3's main key exchange mode.

7.1 Protocol Description

We focus on a slightly simplified version of the handshake encompassing all essential cryptographic aspects for our tightness results. In particular, we only consider mutual authentication and security of the main application traffic keysfirstofone (see [?, ?, ?, ?] for full computational, multi-stage key exchange analyses of the different modes with varying authentication) and accordingly leave out some computations and additional messages. firstofoneTo ease linking back to the underlying SIGMA-I structure, we describe the protocol in the following referencing back to the latter (cf. Section 5). We illustrate the handshake protocol and its accompanying key schedule in Figure 15, the latter deriving keys in the extract-then-expand paradigm of the HKDF key derivation function [?].[7]

In the TLS 1.3 handshake, the client acts as initiator and the server as responder. Within `Hello` messages, both send nonce values n_C resp. n_S together with ephemeral Diffie–Hellman shares g^x resp. g^y. Based on these values, both parties extract a handshake secret HS from the shared DH value DHE $= g^{xy}$ using $\mathsf{HKDF.Extract}$ with a constant salt input.firstoftwo[8] In a second step, client and server derive their respective handshake traffic keys $\mathrm{tk}_{\mathrm{chs}}$, $\mathrm{tk}_{\mathrm{shs}}$ and MAC keys fk_C, fk_S through two levels of $\mathsf{HKDF.Expand}$ steps from the handshake secret HS, including in the first level distinct labels and the hashed communication transcript $\mathsf{H}(\mathtt{CH}\|\mathtt{SH})$ so far as context information.

The handshake traffic keys are then used to encrypt the remaining handshake messages. First the server, then the client send their certificate (carrying their identity and public key), a signature over the hashed transcript up to including their certificatefirstofone ($\mathsf{H}(\mathtt{CH}\|\ldots\|\mathtt{SCRT})$, resp. $\mathsf{H}(\mathtt{CH}\|\ldots\|\mathtt{CCRT})$), as well as a MAC over the (hashed) transcript up to incl. their signaturesfirstofone ($\mathsf{H}(\mathtt{CH}\|\ldots\|\mathtt{SCV})$, resp. $\mathsf{H}(\mathtt{CH}\|\ldots\|\mathtt{CCV})$). Note the similarity to SIGMA-I here: each party signs both nonces and DH values (within $\mathtt{CH}\|\mathtt{SH}$, modulo transcript hashing) together with a unique label, and then MACs both nonces and their own identity (the latter being part of their certificate).firstoftwo[9] The application traffic secret ATS—which we treat as the session key *skey*, unifying secrets of both client and server—is then derived from the master secret MS through $\mathsf{HKDF.Expand}$ with handshake context up to the `ServerFinished` message. The master secret in turn is derived through (context-less) Expand and $\mathsf{Extract}$ from the handshake secret HS.

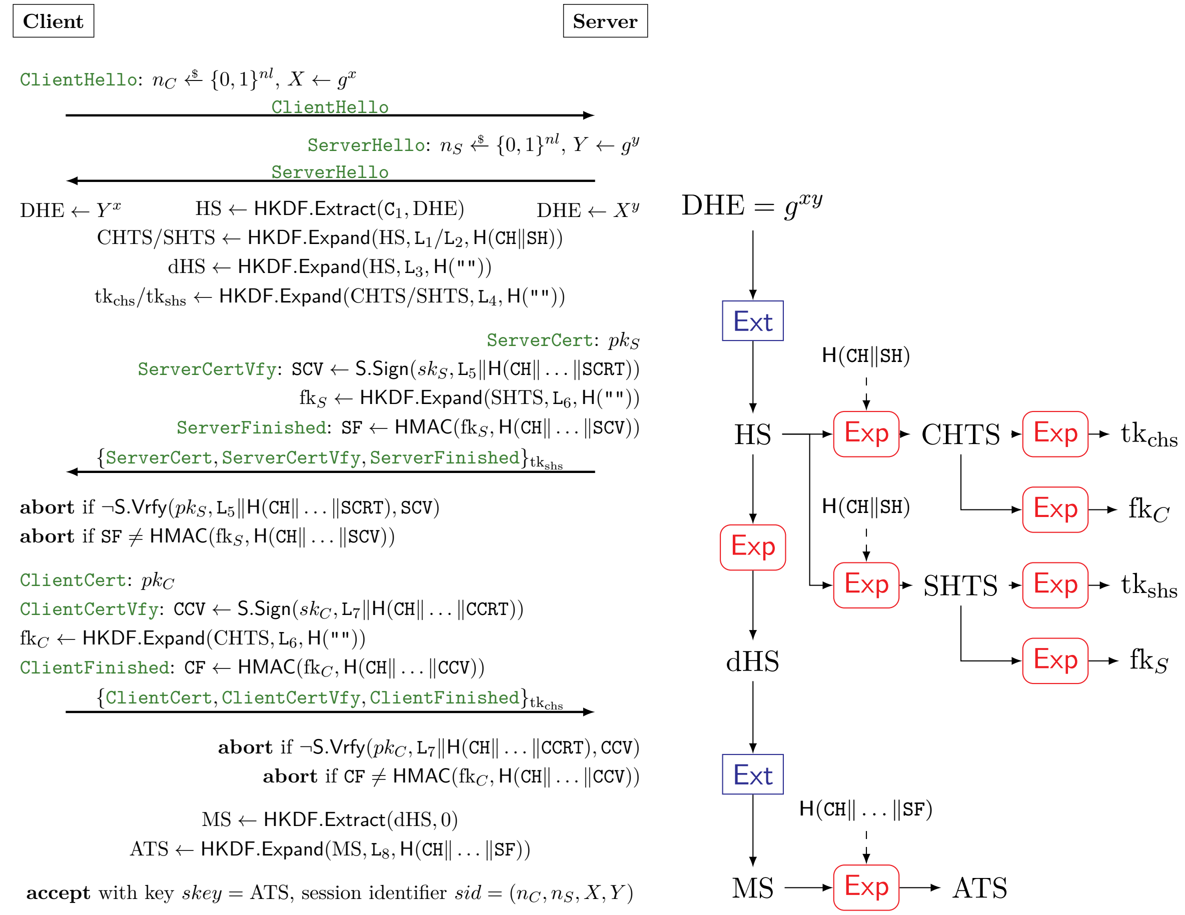
7.2 Handling the TLS 1.3 Key Schedule

As mentioned before, the message flow of the TLS 1.3 handshake relatively closely follows the SIGMA-I design [?, ?] (cf. Figure 7): after exchanging nonces and DH shares (in `Hello`) from

[7] firstofoneWe follow the standard HKDF syntax: $\mathsf{HKDF.Extract}(\mathit{XTS}, \mathit{SKM})$ on input salt *XTS* and source key material *SKM* outputs a pseudorandom key *PRK*. $\mathsf{HKDF.Expand}(\mathit{PRK}, \mathit{CTXinfo})$ on input a pseudorandom key *PRK* and context information *CTXinfo* outputs pseudorandom key material *KM*.

[8] This salt input becomes relevant for pre-shared key handshakes, but in the full handshake takes the constant value $\mathtt{C}_1 = \mathsf{Expand}(\mathsf{Extract}(0, 0), \texttt{"derived"}, \mathsf{H}(\texttt{""}))$.

[9] Instead of using distinct labels for the client and server MAC computations, TLS 1.3 employs distinct MAC keys for client and server, achieving separation between the two MAC values this way.



Client **Server**

$\texttt{ClientHello}$: $n_C \xleftarrow{\$} \{0,1\}^{nl}$, $X \leftarrow g^x$

$\xrightarrow{\texttt{ClientHello}}$

$\texttt{ServerHello}$: $n_S \xleftarrow{\$} \{0,1\}^{nl}$, $Y \leftarrow g^y$

$\xleftarrow{\texttt{ServerHello}}$

$\mathrm{DHE} \leftarrow Y^x$ $\quad \mathrm{HS} \leftarrow \mathsf{HKDF.Extract}(\mathsf{C}_1, \mathrm{DHE})$ $\quad \mathrm{DHE} \leftarrow X^y$

$\mathrm{CHTS}/\mathrm{SHTS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HS}, \mathsf{L}_1/\mathsf{L}_2, \mathsf{H}(\texttt{CH}\|\texttt{SH}))$

$\mathrm{dHS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HS}, \mathsf{L}_3, \mathsf{H}(\texttt{""}))$

$\mathrm{tk_{chs}}/\mathrm{tk_{shs}} \leftarrow \mathsf{HKDF.Expand}(\mathrm{CHTS}/\mathrm{SHTS}, \mathsf{L}_4, \mathsf{H}(\texttt{""}))$

$\texttt{ServerCert}$: pk_S

$\texttt{ServerCertVfy}$: $\texttt{SCV} \leftarrow \mathsf{S.Sign}(sk_S, \mathsf{L}_5\|\mathsf{H}(\texttt{CH}\|\ldots\|\texttt{SCRT}))$

$\mathrm{fk}_S \leftarrow \mathsf{HKDF.Expand}(\mathrm{SHTS}, \mathsf{L}_6, \mathsf{H}(\texttt{""}))$

$\texttt{ServerFinished}$: $\texttt{SF} \leftarrow \mathsf{HMAC}(\mathrm{fk}_S, \mathsf{H}(\texttt{CH}\|\ldots\|\texttt{SCV}))$

$\xleftarrow{\{\texttt{ServerCert}, \texttt{ServerCertVfy}, \texttt{ServerFinished}\}_{\mathrm{tk_{shs}}}}$

abort if $\neg\mathsf{S.Vrfy}(pk_S, \mathsf{L}_5\|\mathsf{H}(\texttt{CH}\|\ldots\|\texttt{SCRT}), \texttt{SCV})$
abort if $\texttt{SF} \neq \mathsf{HMAC}(\mathrm{fk}_S, \mathsf{H}(\texttt{CH}\|\ldots\|\texttt{SCV}))$

$\texttt{ClientCert}$: pk_C
$\texttt{ClientCertVfy}$: $\texttt{CCV} \leftarrow \mathsf{S.Sign}(sk_C, \mathsf{L}_7\|\mathsf{H}(\texttt{CH}\|\ldots\|\texttt{CCRT}))$
$\mathrm{fk}_C \leftarrow \mathsf{HKDF.Expand}(\mathrm{CHTS}, \mathsf{L}_6, \mathsf{H}(\texttt{""}))$
$\texttt{ClientFinished}$: $\texttt{CF} \leftarrow \mathsf{HMAC}(\mathrm{fk}_C, \mathsf{H}(\texttt{CH}\|\ldots\|\texttt{CCV}))$

$\xrightarrow{\{\texttt{ClientCert}, \texttt{ClientCertVfy}, \texttt{ClientFinished}\}_{\mathrm{tk_{chs}}}}$

abort if $\neg\mathsf{S.Vrfy}(pk_C, \mathsf{L}_7\|\mathsf{H}(\texttt{CH}\|\ldots\|\texttt{CCRT}), \texttt{CCV})$
abort if $\texttt{CF} \neq \mathsf{HMAC}(\mathrm{fk}_C, \mathsf{H}(\texttt{CH}\|\ldots\|\texttt{CCV}))$

$\mathrm{MS} \leftarrow \mathsf{HKDF.Extract}(\mathrm{dHS}, 0)$
$\mathrm{ATS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathsf{L}_8, \mathsf{H}(\texttt{CH}\|\ldots\|\texttt{SF}))$

accept with key $skey = \mathrm{ATS}$, session identifier $sid = (n_C, n_S, X, Y)$

Protocol flow legend

$\texttt{MSG}$: Z	message $\texttt{MSG}$ sent, containing Z
$\{\texttt{MSG}\}_K$	message AEAD-encrypted with $K = \mathrm{tk_{shs}}/\mathrm{tk_{chs}}$

Message Abbreviations

CH	ClientHello
SH	ServerHello
CCRT/SCRT	Client/ServerCert
CCV/SCV	Client/ServerCertVfy
CF/SF	Client/ServerFinished

Figure 15: The simplified TLS 1.3 main Diffie–Hellman handshake protocol (left) and key schedule (right). Values L_i and C_i indicate bitstring labels, resp. constant values, (distinct per i). Boxes Ext and Exp denote HKDF extraction resp. expansion, dashed inputs to Exp indicating context information (see protocol figure for detailed computations).

both sides, the remaining (encrypted) messages carry identities (`Certificate`), signatures over the nonces and DH shares (`CertificateVerify`), and MACs over the nonces and identities (`Finished`).

What crucially differentiates the TLS 1.3 handshake from the basic SIGMA-I design (beyond putting more under the respective signatures and MACs, which does not negatively affect the key exchange security we are after) is the way keys are derived. While SIGMA-I immediately derives a master key through a random oracle with input *both* the shared DH secret *and* the session identifying nonces and DH shares, TLS 1.3 separates them in its HKDF-based extract-then-expand key schedule: The core secrets—handshake secret (HS) and master secret (MS)—are derived without further context purely from the shared DH secret $\mathrm{DHE} = g^{xy}$ (beyond other constant inputs). Only when deriving the specific-purpose secrets—handshake traffic keys ($\mathrm{tk_{chs}}$, $\mathrm{tk_{shs}}$), MAC keys (fk_C, fk_S), and session key (ATS)—is context added to the key derivation, including in particular the nonces and DH shares identifying the session. To complicate matters even further, this context is hashed before entering key derivation (or signature and MAC computation), and the final session key ATS depends on more messages than just the session-identifying ones. Since our tighter security proof for the SIGMA(-I) protocol (cf. Section 6) heavily makes use of (exactly) the session identifiers being input together with DH secrets to a random oracle when programming the latter, the question arises how to treat the TLS 1.3 key schedule when aiming at a similar proof strategy.

In their concurrent work, Diemert and Jager [?] satisfy this requirement by modeling the full derivation of each stage key in their multi-stage treatment as a separate random oracle. This directly connects inputs to keys, but results in a monolithic random oracle treatment of the key schedule which loses the independence of the intermediate HKDF.Extract and HKDF.Expand steps in translation.

We overcome the technical obstacle of this linking while staying closer to the structure of TLS 1.3's key schedule. First of all, we directly model both HKDF.Extract and HKDF.Expand as individual (programmable) random oracles, which leads to a slightly less excessive use of the random oracle technique. We then have to carefully orchestrate the programming of intermediate secrets and session keys in a two-level approach, connecting them through constant-time look-ups, and taking into account that inputs to deriving the session keys depend on values established through the intermediate secrets (namely, the server's `Finished` MAC). Along the way, we separately ensure that we recognize any hashed inputs of interest that the adversary might query to the random oracle, without modeling the hash function H as a random oracle itself. By tracking intermediate programming points (especially HS and MS) in the random oracles, we recover the needed capability of linking sessions and their session identifiers and DH shares exchanged to the corresponding session keys. This finally allows us to again (efficiently) determine when and on what input to query the strong Diffie–Hellman oracle when programming challenge DH shares into the TLS 1.3 key exchange execution during the proof.

8 Tighter Security Proof for the TLS 1.3 Handshake

We now give our second main result, the tighter-security bound firstoftwofor the TLS 1.3 handshake protocolfor TLS 1.3.

Theorem 8.1. *Let $\mathcal{A}$ be a key exchange security adversary against the TLS 1.3 handshake protocol as specified in Figure 15 based on a hash function* H*, a signature scheme* S*, and a group* $\mathbb{G}$ *of prime order* p*, and let the* HKDF *functions* Extract *and* Expand *in the protocol be modeled as (independent) random oracles* RO_1*, resp.* RO_2*. For any* $(t, q_{\mathrm{N}}, q_{\mathrm{S}}, q_{\mathrm{RS}}, q_{\mathrm{RL}}, q_{\mathrm{T}})$-KE-SEC*-adversary against SIGMA-I making at most* q_{RO} *queries to the random oracle, we give algorithms* $\mathcal{B}_1$*,* $\mathcal{B}_2$*,* $\mathcal{B}_3$*, and* $\mathcal{B}_4$ *in the proof, with running times* $t_{\mathcal{B}_i} \approx t$ *(for* $i = 1, 3, 4$*) and* $t_{\mathcal{B}_2} \approx t + 2q_{\mathrm{RO}} \log_2 p$ *close to that of* $\mathcal{A}$*, such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{KE\text{-}SEC}}_{\mathrm{TLS\,1.3}}(t, q_{\mathrm{N}}, q_{\mathrm{S}}, q_{\mathrm{RS}}, q_{\mathrm{RL}}, q_{\mathrm{T}}) &\le \frac{3q_{\mathrm{S}}^2}{2^{nl+1}\cdot p} + \mathsf{Adv}^{\mathsf{CR}}_{\mathsf{H}}(t_{\mathcal{B}_1}) \\ &+ 2\cdot \mathsf{Adv}^{\mathsf{stDH}}_{\mathbb{G}}(t_{\mathcal{B}_2}, q_{\mathrm{RO}}) + \frac{q_{\mathrm{RO}}\cdot q_{\mathrm{S}}}{2^{kl-1}} + \mathsf{Adv}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{S}}(t_{\mathcal{B}_3}, q_{\mathrm{N}}, q_{\mathrm{S}}, q_{\mathrm{S}}, q_{\mathrm{RL}}) \\ &+ \mathsf{Adv}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{HMAC}}(t_{\mathcal{B}_4}, q_{\mathrm{S}}, q_{\mathrm{S}}, 1, q_{\mathrm{S}}, 1, 0).\end{aligned}$$

Here, $nl = 256$ *is the nonce length in TLS 1.3,* kl *is the output length of* $\mathsf{RO}_2 = \mathsf{HKDF.Expand}$*,* $\mathbb{G}$ *is the used Diffie–Hellman group of prime order* p*, and* $q_{\mathrm{S}} \cdot q_{\mathrm{RO}} \le 2^{kl-3}$.[10]

[10] We simplify the factor on $\mathsf{Adv}^{\mathsf{stDH}}_{\mathbb{G}}$ to 2 by assuming $q_{\mathrm{S}} \cdot q_{\mathrm{RO}} \le 2^{kl-3}$, which is true for any reasonable real-world parameters. See the proof for the exact bound.

Proof. We prove our bound by making an incremental series of changes to the key exchange security game and limiting the amount that each change affects the success probability of $\mathcal{A}$.

Game 0. The initial game, Game G_0, is the key exchange security game for TLS played by $\mathcal{A}$, using the implicit KGen, $\mathsf{Activate}$, and Run routines defined by the TLS protocol specification on the left side of Figure 15. (In this game, $\mathsf{HKDF.Extract}$ and $\mathsf{HKDF.Expand}$ are modeled by random oracles RO_1 and RO_2 respectively.) By definition,

$$\Pr[\mathrm{G}_0 \Rightarrow 1] = \Pr[\mathrm{G}^{\mathsf{KE\text{-}SEC}}_{\mathsf{TLS},\mathcal{A}} \Rightarrow 1].$$

Game 1. In game G_0, we start logging the nonces and group elements chosen by honest sessions. Whenever two honest sessions choose the same nonces or group elements, we set a flag bad_C. Whenever an honest responder session chooses a nonce and group element that have already been received by another session, we set a flag bad_O. We also make both random oracles RO_1 and RO_2 lazily sampled using internal tables H_1 and H_2. These changes only affect the values of the game's internal state, and the view of the adversary remains the same as in G_0, so

$$\Pr[\mathrm{G}_1 \Rightarrow 1] = \Pr[\mathrm{G}_0 \Rightarrow 1].$$

Game 2. Starting with G_2, we abort whenever two honest sessions sample the same nonce or group element and whenever an honest responder samples a nonce and group element that are already in use. Since this happens only after one of the flags bad_C and bad_O is set, by the identical-until-bad lemma,

$$\Pr[\mathrm{G}_1 \Rightarrow 1] - \Pr[\mathrm{G}_2 \Rightarrow 1] \leq \Pr[\mathsf{bad}_C \leftarrow \mathsf{true} \text{ or } \mathsf{bad}_O \leftarrow \mathsf{true} \text{ in } \mathrm{G}_1].$$

One nonce and one group element is chosen in each $\mathsf{RunInit1}$ call and each $\mathsf{RunResp1}$ call, so at most one nonce and one group element is chosen for each of the q_{S} queries the adversary makes to its $\textsc{Send}$ oracle. We use the birthday bound to limit the probability of a collision (flag bad_C) in either the set of honest sessions' nonces or the set of honest sessions' DH shares to $\frac{q_{\mathrm{S}}^2}{2^{nl+1} \cdot p}$. Every time a responder session chooses a nonce and group element, there are at most q_{S} values have already been chosen, so by the union bound bad_O is set with probability at most $\frac{q_{\mathrm{S}}^2}{2^{nl} \cdot p}$. Therefore

$$\Pr[\mathrm{G}_1 \Rightarrow 1] - \Pr[\mathrm{G}_2 \Rightarrow 1] \leq \frac{3q_{\mathrm{S}}^2}{2^{nl+1} \cdot p}.$$

Game 3. Next, we must ensure that partial transcripts between honest sessions do not collide under the hash function H. This is a step unique to the TLS proof, which hashes all of its context with a collision-resistant hash function before it is input into key-derivation. In G_3, honest sessions will log all of their hash outputs in a look-up table T: whenever an honest session computes $d = \mathsf{H}(s)$ for some string s, it sets $T[d] \leftarrow s$ if $T[d]$ has not already been defined. If $T[d]$ is not empty, then some prior honest session has computed $d = \mathsf{H}(s')$ for some string s'. The session will set a flag $\mathsf{bad}_{\mathsf{H}}$ if $s' \neq s$, noting that a collision has occurred. We also remove the now superfluous bad_C flag. These administrative changes do not affect the view of the adversary, so

$$\Pr[\mathrm{G}_3 \Rightarrow 1] = \Pr[\mathrm{G}_2 \Rightarrow 1].$$

Game 4. In Game G_4, we abort whenever hashes computed by honest sessions collide (i.e. the $\mathsf{bad_H}$ flag is set). By the identical-until-bad lemma,

$$\Pr[G_3 \Rightarrow 1] - \Pr[G_4 \Rightarrow 1] \leq \Pr[\mathsf{bad_H} \leftarrow \mathsf{true} \text{ in } G_3].$$

We bound the probability that $\mathsf{bad_H}$ is set via a reduction $\mathcal{B}_1$ to the collision-resistance security of H. The reduction simulates G_3 honestly for the adversary $\mathcal{A}$. If the flag $\mathsf{bad_H}$ is set, then the reduction has obtained strings s, s', and d such that $s' \neq s$, and $\mathsf{H}(s) = \mathsf{H}(s') = d$. Then $\mathcal{B}_1$ outputs (s, s') and wins the collision-resistance game, so $\mathsf{Adv}^{\mathsf{cr}}_{\mathsf{H},\mathcal{B}_1} \geq \Pr[\mathsf{bad_H} \leftarrow \mathsf{true} \text{ in } G_3]$. The runtime $t_{\mathcal{B}_1}$ of $\mathcal{B}_1$ approximately equals the runtime of $\mathcal{A}$ in G_3. It follows that

$$\Pr[G_3 \Rightarrow 1] - \Pr[G_4 \Rightarrow 1] \leq \mathsf{Adv}^{\mathsf{CR}}_{\mathsf{H}}(t_{\mathcal{B}_1}).$$

Game 5. In Game G_5, we remove the superfluous $\mathsf{bad_H}$ flag and make additional internal changes to the behavior of honest sessions. As in the SIGMA-I proof, all honest initatior sessions now log the first message they send in a set Sent, and honest responder sessions use this set to check whether their first received message came from an honest session without tampering. If so, we say the responder session has an "honest origin partner." In the SIGMA-I protocol, partnering between honest sessions was sufficient to ensure agreement on the derived master key and all subsequently computed keys, since partners are guaranteed to hold the same nonces and group elements. In TLS 1.3, partnering also ensures agreement on the handshake traffic secrets SHTS and CHTS, but it does not ensure agreement on the session key ATS. Therefore the responder only logs the handshake traffic keys $\mathrm{fk}_S, \mathrm{fk}_C, \mathrm{tk}_{\mathrm{shs}}$, and $\mathrm{tk}_{\mathrm{chs}}$ in a look-up table S under its session identifier. In addition to the session identifier, the application traffic secret ATS depends on the server's identity $\mathtt{SCRT}$, signature $\mathtt{SCV}$, and MAC tag $\mathtt{SF}$. These values are not necessarily shared by partner sessions in Game G_5, so two partnered sessions may derive different values of ATS. The responder session therefore logs its session key ATS in a second look-up table S' indexed by all of the dependencies of the session key: $sid, \mathtt{SCRT}, \mathtt{SCV}$, and $\mathtt{SF}$. All of this is just bookkeeping, so

$$\Pr[G_5 \Rightarrow 1] = \Pr[G_4 \Rightarrow 1].$$

Game 6. Going forward from Game G_6, honest initiators copy their key material from tables S and S' if it is consistent for them to do so. In the case where the adversary has tampered with the values of $\mathtt{SCRT}$, $\mathtt{SCV}$, or $\mathtt{SF}$, the partner's session key depends on the untampered values and should not be copied. Therefore honest initiators always copy encryption and MAC keys from the table S if they have an honest partner session, but they only copy ATS when the $\mathtt{SCRT}, \mathtt{SCV}$, and $\mathtt{SF}$ messages they received match the ones sent by their partner. The initiator session can check whether tampering occurred using the table S', which will contain a session key ATS at index $sid \| \mathtt{SCRT} \| \mathtt{SCV} \| \mathtt{SF}$ if and only if the honest partner session computed and sent $\mathtt{SCRT}$, $\mathtt{SCV}$, and $\mathtt{SF}$.

We argue that all copied keys are consistent with the keys that would be derived in G_5. Recall that partnered sessions agree on the nonces and the DH shares X and Y as components of sid, so they also agree on the shared DH secret Z associated with the pair (X, Y). Partnered sessions therefore agree on the handshake secret HS, which is derived from Z without context, and on the handshake traffic secrets, which are derived with the session identifier as context. Thus partnered sessions agree on the values of the handshake traffic keys $\mathrm{fk}_S, \mathrm{fk}_C, \mathrm{tk}_{\mathrm{shs}}$, and $\mathrm{tk}_{\mathrm{chs}}$ which are derived from the handshake traffic secrets. For the adversary it is hence unobservable if honest sessions compute the handshake traffic keys themselves, or copy the keys from their partners. By agreeing on the handshake secret HS, partnered sessions will also agree on the master secret MS, which is derived from HS without context. The if $\texttt{SCRT}$, $\texttt{SCV}$, and $\texttt{SF}$ are left untampered, both sessions will derive the session key as $\mathsf{RO}_2(\mathrm{MS}, \texttt{L}_8, \mathsf{H}(\mathit{sid} \| \texttt{SCRT} \| \texttt{SCV} \| \texttt{SF})])$. Hence it is again unobservable whether an honest initiator derives ATS itself or copies ATS from an honest partner which agrees on the values of $\texttt{SCRT}$, $\texttt{SCV}, \texttt{SF}$; consequently

$$\Pr[\mathrm{G}_6 \Rightarrow 1] = \Pr[\mathrm{G}_5 \Rightarrow 1].$$

Game 7. In Game G_7, all responders sample ATS, SHTS and CHTS randomly (unless their values have already been fixed by queries to random oracle RO_2 on the corresponding input), then retroactively programs random oracle RO_2 by setting its internal table H_2 on the appropriate input. Partnered initiator sessions which have not copied ATS (i.e., those who received tampered $\texttt{SCRT}$, $\texttt{SCV}$, and $\texttt{SF}$) also sample ATS randomly and program RO_2 when necessary. We choose to program ATS, SHTS, and CHTS, as opposed to only mk in the SIGMA-I proof, because these three keys are derived with context. Most importantly, the DH shares X and Y indirectly enter the key derivation for these keys, which will be critical for the reduction in the next step. This simply moves the lazy sampling process from RO_2 to $\mathsf{RunResp1}$ and $\mathsf{RunInit2}$ for certain queries, which is unobservable to the adversary; therefore

$$\Pr[\mathrm{G}_7 \Rightarrow 1] = \Pr[\mathrm{G}_6 \Rightarrow 1].$$

Game 8. The step between G_7 and G_8 is most technically involved step of this proof, and it is also the most significantly altered from the corresponding step in the proof of SIGMA-I. In G_8, partnered initiators and responder sessions with honest origin partners will stop maintaining the consistency of their keys ATS, SHTS, and CHTS with the random oracle RO_2. Specifically, responders with honest origin partners sample ATS, SHTS, and CHTS uniformly at random even if RO_2 has already been queried on the string $\mathrm{HS}, \texttt{L}, d$ for the appropriate label and hash, and they do not retroactively program RO_2. Partnered initiator sessions which have not copied ATS from their partner also sample ATS uniformly without checking or programming RO_2. These keys are therefore completely random, and they will be inconsistent with any random oracle queries made before or after the keys are sampled.

In order to detect this inconsistency, the adversary must make a query to RO_2 that would, in G_7, return one of the unprogrammed keys. Which queries are these? They are the queries that an honest responder session with honest origin partner would use to derive SHTS, CHTS, and ATS, and the queries that an honest partnered initiator which received a tampered message would use to derive ATS. Formally, let $\mathit{sid} = (n, n', X, Y)$ be the session ID held by some honest responder session with honest origin partner, and let $\texttt{SCRT}$, $\texttt{SCV}$, $\texttt{SF}$ be the identity, signature, and MAC tag sent by this session. Let DHE be the DH secret corresponding to the pair (X, Y). Then the adversary $\mathcal{A}$ can detect an inconsistency (in derviations of honest responders) in game G_8 if at any point during the game $\mathcal{A}$ queries RO_2 on one of the tuples

$$(\mathsf{RO}_1(\texttt{C}_1, \mathrm{DHE}), \texttt{L}, \mathsf{H}(\mathit{sid})) \qquad \text{or} \qquad (\mathrm{MS}, \texttt{L}_8, \mathsf{H}(\mathit{sid} \| \texttt{SCRT} \| \texttt{SCV} \| \texttt{SF})),$$

where $\texttt{L} \in \{\texttt{L}_1, \texttt{L}_2\}$ and where for some HS, dHS, we have that $\mathrm{HS} = \mathsf{RO}_1(\texttt{C}_1, \mathrm{DHE})$, that $\mathrm{dHS} = \mathsf{RO}_2(\mathrm{HS}, \texttt{L}_3, \mathsf{H}(\texttt{""}))$, and that $\mathrm{MS} = \mathsf{RO}_1(\mathrm{dHS}, 0)$. Otherwise (for derviations of honest initiators), let sid be the session ID held by an honest partnered initiator session, and let $\texttt{SCRT}$, $\texttt{SCV}$, and $\texttt{SF}$ be the identity, signature, and MAC tag received by that session. For initiator sessions that do not copy ATS, at least one of these values was not sent by the honest partner. Then the adversary $\mathcal{A}$ can detect an inconsistency in game G_8 if at any point it queries RO_2 on the tuple

$$(\mathrm{MS}, \texttt{L}_8, \mathsf{H}(\mathit{sid} \| \texttt{SCRT} \| \texttt{SCV} \| \texttt{SF})),$$

where for some HS, dHS, we have that $\text{HS} = \mathsf{RO}_1(\mathtt{C}_1, \text{DHE})$, that $\text{dHS} = \mathsf{RO}_2(\text{HS}, \mathtt{L}_3, \mathsf{H}(\texttt{""}))$, and that $\text{MS} = \mathsf{RO}_1(\text{dHS}, 0)$. Let event F denote the event that the adversary $\mathcal{A}$ makes at least one of the above queries. If event F does not occur, then ATS, SHTS, and CHTS are chosen uniformly at random in both G_7 and G_8, hence

$$\Pr[\text{G}_7 \Rightarrow 1] - \Pr[\text{G}_8] \Rightarrow 1] \leq \Pr[F \text{ occurs in } \text{G}_7].$$

We bound the probability of event F via a reduction $\mathcal{B}_2$ to the strong Diffie–Hellman assumption in group $\mathbb{G}$. The reduction will make no more queries to its stDH oracle than $\mathcal{A}$ makes to its RO_2 oracles.

Given its strong DH challenge ($A = g^a, B = g^b$) and having access to the strong Diffie–Hellman oracle stDH_a, $\mathcal{B}_2$ simulates G_7 for an adversary $\mathcal{A}$ in the following manner: In all honest initiator sessions, $\mathcal{B}_2$ samples r uniformly at random from $\mathbb{Z}_p$ and sets the session's DH share $X \leftarrow A \cdot g^r$. In all honest responder sessions with honest origin partner, $\mathcal{B}_2$ samples r' uniformly from $\mathbb{Z}_p$ and sets the session's DH share $Y \leftarrow B \cdot g^{r'}$. Both of these DH shares are still distributed uniformly over $\mathbb{Z}_p$ as long as p is prime and A and B are not the identity. To extract g^{ab} when event F occurs, the reduction $\mathcal{B}_2$ will follow the same general strategy as the reduction $\mathcal{B}_1$ in the proof of SIGMA-I, with four major points of divergence. We address these points first, before giving a full description of $\mathcal{B}_2$.

1. Since $\mathcal{B}_2$ no longer knows x or y such that $X = g^x$ or $Y = g^y$, it cannot compute the Diffie–Hellman secret DHE or the derived handshake secret HS, so it samples HS randomly for honest responder sessions with honest origin partners and for honest partnered initiator sessions. The adversary can only tell that HS was not correctly computed if it notices that SHTS, CHTS, or dHS are derived from an incorrect value of HS. The former two cases require the adversary to make a query that triggers event F. In the latter case, dHS is not revealed to the adversary through any oracle, so the adversary must notice that ATS, which is derived indirectly from dHS via the master secret, is derived from an incorrect value of HS. This also requires $\mathcal{A}$ to make a query that triggers event F. Therefore, until event F occurs, this change is unobservable to the adversary.

2. In the TLS protocol, the context string, including the Diffie–Hellman shares X and Y, is hashed with H before it enters key derivation, so $\mathcal{B}_2$ cannot directly associate a query to RO_2 with the honest session(s) whose session ID is being used. The reduction addresses this by having each honest responder with honest origin partner and each honest partnered initiator, log the hash of its context in a reverse look-up table R. (The context does not include the handshake or master secrets.) Then in the RO_2 oracle, $\mathcal{B}_2$ can use R to efficiently check whether the hash d of a query is used to derive a handshake or application traffic key.

3. Due to TLS's complex key schedule, no one random oracle query contains both a pair of Diffie–Hellman shares and the DH secret associated with that pair. Instead, $\mathcal{B}_2$ will augment the RO_1 and RO_2 oracles to log in a reverse look-up table T the DH secret associated with each of the intermediate values HS, dHS, and MS. The DH secret for $\text{dHS} = \mathsf{RO}_2(\text{HS}, \mathtt{L}_3, \mathsf{H}(\texttt{""}))$ simply be copied from $T[\text{HS}]$, and the DH secret for $\text{MS} = \mathsf{RO}_1(\text{dHS}, 0)$ will be copied from $T[\text{dHS}]$. For each query to RO_2 with secret s, the reduction can efficiently check using T whether s was derived from some DH secret via RO_1.

4. The TLS key schedule uses multiple random oracle queries (if we model $\mathsf{HKDF.Extract}$ and $\mathsf{HKDF.Expand}$ as random oracles) whereas the SIGMA-I protocol uses only one. If $\mathcal{A}$ can guess the intermediate value $\text{HS} = \mathsf{RO}_1(\mathtt{C}_1, \text{DHE})$, where DHE is the DH secret associated to some

pair of embedded shares (X, Y) chosen by honest sessions, then it can trigger event F without ever submitting DHE to an oracle. In this case, $\mathcal{A}$ can trigger event F, but $\mathcal{B}_2$ cannot win the Strong DH game. However, if $\mathsf{RO}_1(\mathtt{C}_1, \mathrm{DHE})$ is never queried, then it is uniformly random, and the probability that $\mathcal{A}$ guesses correctly is bounded by $\frac{q_{\mathrm{RO}} \cdot q_{\mathrm{S}}}{2^{kl}}$ by the birthday bound.

To compute the correct handshake and application traffic keys, $\mathcal{B}_2$ needs to be able to correctly program CHTS, SHTS, and ATS. When these keys are chosen by an honest responder with honest origin partner or a partnered initiator, $\mathcal{B}_2$ uses its strong DH oracle to check whether RO_2 has already received the query that the adversary needs to make to generate these keys. If the query has already been made, $\mathcal{B}_2$ can look up the DH secret using T and win the game. Otherwise, $\mathcal{B}_2$ hashes the session's context and logs it in R, so that future RO_2 queries can identify this session for retroactive programming. It also logs the session's randomness in a look-up table Q, to be used if event F is triggered relative to this session by a future RO_2 query.

Like in the SIGMA-I proof, $\mathcal{B}_2$ must be able to correctly compute handshake and application traffic keys for unpartnered initiator sessions. Because all initiator sessions have embedded DH shares, $\mathcal{B}_2$ cannot compute the DH secret DHE for these sessions. However, it can use its StrongDH oracle to check whether the adversary has queried such a secret and copy the expected keys to preserve consistency in this case. If no query has been made, the keys are selected randomly and the initiator session stores its context, randomness, and keys in R. In future queries to the RO_2 oracle, $\mathcal{B}_2$ will use R to efficiently check whether a query should output one of the initiator session's keys. If so, it retroactively programs the oracle using the keys from R.

Therefore, if event F occurs, reduction $\mathcal{B}_2$ wins the strong Diffie–Hellman game except with probability $\frac{q_{\mathrm{RO}} \cdot q_{\mathrm{S}}}{2^{kl}}$, resulting in $\mathsf{Adv}^{\mathsf{stDH}}_{\mathbb{G}}(t_{\mathcal{B}_2}, q_{\mathrm{RO}}) \geq (1 - \frac{q_{\mathrm{RO}} \cdot q_{\mathrm{S}}}{2^{kl}}) \cdot \Pr[F]$. Then $\Pr[F] \leq \frac{2^{kl}}{2^{kl} - q_{\mathrm{RO}} \cdot q_{\mathrm{S}}} \cdot \mathsf{Adv}^{\mathsf{stDH}}_{\mathbb{G}}(t_{\mathcal{B}_2}, q_{\mathrm{RO}})$. Otherwise, the reduction simulates G_7 perfectly except with probability $\frac{q_{\mathrm{RO}} \cdot q_{\mathrm{S}}}{2^{kl}}$.

$$\begin{aligned}
\Pr[\mathrm{G}_7 \Rightarrow 1] &= \Pr[\mathrm{G}_8 \Rightarrow 1] + \Pr[F] + (1 - \Pr[F]) \cdot \frac{q_{\mathrm{RO}} \cdot q_{\mathrm{S}}}{2^{kl}} \\
&\leq \Pr[\mathrm{G}_8 \Rightarrow 1] + \frac{2^{kl} + q_{\mathrm{RO}} \cdot q_{\mathrm{S}}}{2^{kl} - q_{\mathrm{RO}} \cdot q_{\mathrm{S}}} \cdot \mathsf{Adv}^{\mathsf{stDH}}_{\mathbb{G}}(t_{\mathcal{B}_2}, q_{\mathrm{RO}}) + \frac{q_{\mathrm{RO}} \cdot q_{\mathrm{S}}}{2^{kl}} \\
&\leq \Pr[\mathrm{G}_8 \Rightarrow 1] + 2 \cdot \mathsf{Adv}^{\mathsf{stDH}}_{\mathbb{G}}(t_{\mathcal{B}_2}, q_{\mathrm{RO}}) + \frac{q_{\mathrm{RO}} \cdot q_{\mathrm{S}}}{2^{kl}},
\end{aligned}$$

where the last simplification step assumes that $q_{\mathrm{S}} \cdot q_{\mathrm{RO}} \leq 2^{kl-2}$, which is true for any reasonable real-world parameters.

Game 9. In Game G_9, honest responders with honest origin partners sample fk_S, fk_C, $\mathrm{tk}_{\mathrm{chs}}$ and $\mathrm{tk}_{\mathrm{shs}}$ uniformly at random, so these keys are no longer consistent with RO_2. The adversary can distinguish this change if and only if it queries RO_2 on a string $\mathrm{SHTS}, \mathtt{L}, \mathsf{H}(\texttt{""})$, or $\mathrm{CHTS}, \mathtt{L}, \mathsf{H}(\texttt{""})$, where $\mathtt{L} \in \{\mathtt{L}_4, \mathtt{L}_6\}$, and SHTS and CHTS are chosen by an honest responder sessions with honest origin partner. Call this event E. In these sessions, SHTS and CHTS are chosen uniformly at random by G_8, and they are never revealed by any oracle. Therefore the probability of event E is at most $\frac{q_{\mathrm{RO}} \cdot q_{\mathrm{S}}}{2^{kl}}$ by the birthday bound, hence

$$\Pr[\mathrm{G}_8] \leq \Pr[\mathrm{G}_9] + \frac{q_{\mathrm{RO}} \cdot q_{\mathrm{S}}}{2^{kl}}.$$

Note that this step in the SIGMA-I proof introduced a multi-user PRF security bound due to final keys being derived through a PRF, not the random oracle. Modeling $\mathsf{HKDF.Expand}$ as random oracle RO_2, we here instead incur a birthday bound under the random oracle instead of a multi-user PRF security bound for $\mathsf{HKDF.Expand}$.

The remaining game hops are identical to those in the proof of SIGMA-I, so we discuss them only briefly.

Game 10. In Game G_{10}, we log all messages signed by an honest session in a look-up table Q_S, and we set a flag bad_S whenever a partnered session verifies a signature with an uncorrupted public key on a message that was not in Q_S. This is just administrative, so

$$\Pr[\mathrm{G}_{10} \Rightarrow 1] = \Pr[\mathrm{G}_9 \Rightarrow 1].$$

Game 11. In Game G_{11}, we abort if the flag bad_S is set. In this case, an honest partnered session received a signature which was not computed by an honest session, and which was verified by an uncorrupted public key. We can give a straightforward reduction $\mathcal{B}_3$ to the multi-user EUF-CMA security of the signature scheme that wins whenever bad_S is set and has runtime approximately equal to that of $\mathcal{A}$ in G_{10}. By the identical-until-bad lemma,

$$Pr[\mathrm{G}_{10} \Rightarrow 1] - \Pr[\mathrm{G}_{11} \Rightarrow 1] \leq \mathsf{Adv}_{\mathsf{S}}^{\mathsf{mu\text{-}EUF\text{-}CMA}}(t_{\mathcal{B}_3}, q_{\mathrm{N}}, q_{\mathrm{S}}, q_{\mathrm{S}}, q_{\mathrm{RL}}).$$

Interestingly and in contrast to the SIGMA-I proof, soundness is still not guaranteed after this game hop, because we do not require the signature scheme to be strongly unforgeable. Therefore the adversary may be able to produce a new signature on a message that had been signed by an honest session, allowing it to tamper with $\mathtt{SCV}$ without setting the bad_S flag.

Game 12. In Game G_{12} we log all messages for which an honest session computed a MAC tag in a look-up table Q_M. We remove the bad_S flag and instead set a flag bad_M if an honest partnered session verifies a MAC on a message that is not in Q_M. Again, this is only bookkeeping and does not impact the view of $\mathcal{A}$, hence

$$\Pr[\mathrm{G}_{12} \Rightarrow 1] = \Pr[\mathrm{G}_{11} \Rightarrow 1].$$

Game 13. Finally, in Game G_{13}, we abort if an honest session with an honest partner verifies a MAC tag on a message which was not tagged by any honest session; i.e if the bad_M flag is set. We can give a simple reduction $\mathcal{B}_4$ to multi-user MAC security. The reduction $\mathcal{B}_4$ assigns a pair of indices $i, i+1$ to each session identifier held by an honest session with honest origin partner. When an honest session with honest origin partner needs to compute a server MAC tag, $\mathcal{B}_4$ finds the pair $(i, i+1)$ using the session identifier and calls its Tag oracle with user identity i. When the session needs a client MAC tag $\mathcal{B}_4$ calls Tag with user identity $i+1$. The reduction calls its Tag oracle no more than twice for every query $\mathcal{A}$ makes to Send (once to generate a tag, and once to verify a tag). Since by Game G_9 all honest sessions with honest origin partners sample their MAC keys fk_S and fk_C uniformly at random, the keys implicitly generated by the MAC security game are consistent with the expected operation of Game G_{13}. When the flag bad_M is set, a partnered session has received a valid tag on a message which was never logged in Q_M. The reduction can look up the pair $(i, i+1)$ using the session identifier of whichever session set bad_M. Since $\mathcal{B}_4$ logs every message for which it calls its Tag oracle, this is a valid forgery for either identity i or identity $i+1$, and $\mathcal{B}_4$ will win. Then

$$\Pr[\mathrm{G}_{12} \Rightarrow 1] - \Pr[\mathrm{G}_{13} \Rightarrow 1] \leq \mathsf{Adv}_{\mathsf{M}}^{\mathsf{mu\text{-}EUF\text{-}CMA}}(t_{\mathcal{B}_4}, q_{\mathrm{S}}, q_{\mathrm{S}}, 1, q_{\mathrm{S}}, 1, 0).$$

The runtime of $\mathcal{B}_4$ is about that of $\mathcal{A}$ in G_{12}.

We can now finally argue that the advantage of $\mathcal{A}$ in G_{13} is zero. The adversary $\mathcal{A}$ would win game G_{13} with probability better than $\frac{1}{2}$ in one of three ways:

1. Sound is false,
2. $\mathsf{ExplicitAuth}$ is false, or
3. Fresh is true and $b' = b$.

Soundness. The flag Sound is set if (1) three honest sessions hold the same session identifier, or if (2) two partnered sessions accept with different session keys. By Game G_2, each session identifier is held by at most one session of each role. There are only two roles so case (1) never occurs. If two partnered sessions π_1 and π_2 accept, the initiator session π_1 verified a MAC tag τ on the message $m = n\|n'\|X\|Y\|\mathtt{SCRT}\|\mathtt{SCV}$. Because τ was verified by an honsest partnered session, by Game G_{13}, this message was tagged by an honest session. Honest sessions only tag strings including their own nonce, and by Game G_2, the only honest session with nonce n' is π_2. Then π_2 must have tagged the message m, so π_1 and π_2 agree on both τ and m. Since the DH shares X and Y are components of m, π_1 and π_2 also agree on the DH secret DHE associated with the pair (X, Y). Consequently, π_1 and π_2 will agree on any value derived deterministically from m, τ, and DHE, including the session key ATS. Then the flag Sound is always true in G_{13}.

Explicit authentication. The flag $\mathsf{ExplicitAuth}$ is set if there exists a session π_u^i that accepts with uncorrupted peer identity v, and either (1) no honest session π_v^j is partnered with π_u^i, or (2) a session π_v^j is partnered with π_u^i but accepts with peer identity $w \neq u$. To have accepted with peer identity v, the session π_u^i must have received and verified a signature σ using the public key of identity v on a message m containing the session identifier of π_u^i. As v was uncorrupted at the time that π_u^i accepted, by Game G_{11}, the message m must have been signed by some honest session π_v^j. As honest sessions only sign messages containing their own session identifiers, $\pi_v^j.sid = \pi_u^i.sid$, so π_v^j and π_u^i are partnered. If case (2) occurs, π_v^j must have accepted a MAC tag τ on message m' containing its session ID and the identity w of its peer. We know that π_v^j is a partnered session, so by G_{13}, m' was tagged by some honest session. Honest sessions tag only messages containing their own session identifiers, so by G_2, the message m' must have been tagged by either π_u^i or π_v^j. In SIGMA-I, the messages tagged by these two sessions are differentiated by there labels. Here, they are differentiated by their length: one role signs a message including values $\mathtt{SF}$, $\mathtt{CCRT}$, and $\mathtt{CCV}$, while the other signs a message which does not contain these values. For this reason π_v^j will not verify the tag on a message it signed itself. Therefore m' must have been tagged by π_u^i, so m' contains the identity u. This contradicts the assumption that $w \neq u$, so case (2) never occurs, and the flag $\mathsf{ExplicitAuth}$ is always false in G_{13}.

Guessing the challenge bit. Now the adversary can only win with advantage better than zero is by guessing the correct value of b when the Fresh flag is set to true. This requirement ensures that all tested sessions accepted with uncorrupted peer identities. Since $\mathsf{ExplicitAuth}$ is true, each tested session must therefore have an honest session with which it is partnered, and by Sound, this session holds the same session key. Then by G_6, each tested initiator session copies the session key of its partner. By G_8 each tested responder session, and each responder session partnered with a tested initiator session chooses its session key uniformly at random. By Fresh, the partners of tested sessions were not tested or revealed. Then the session keys of all tested sessions are sampled uniformly and never revealed to the adversary by any oracle. Therefore the key returned by each $\textsc{Test}$ query is uniformly random and independent of the bit b. The adversary's view is independent of the bit b, so it will win G_{13} with probability $\frac{1}{2}$, and consequently its advantage is 0.

Collecting the bounds across all game hops gives the theorem statement. $\square$

9 Evaluation

Tighter security results in terms of loss factors are practically meaningful only if they materialize in better concrete advantage bounds when taking the underlying assumptions into account. In our case, this amounts to the question: How does the overall concrete security of the SIGMA/SIGMA-I and the TLS 1.3 key exchange protocols improve based on our tighter security proofs?

Parameter selection. In order to evaluate our and prior bounds pratically, we need to make concrete choices for each of the parameters entering the bounds. Let us explain the choices we made in our evaluation:

Runtime $t \in \{2^{40}, 2^{60}, 2^{80}\}$. We parameterize the adversary's runtime between well within computational reach (2^{40}) and large-scale attackers (2^{80}).

Number of users $\#U = q_\mathrm{N} \in \{2^{20}, 2^{30}\}$. We consider the number of users a global-scale adversary may interact with to be in the order of active public-key certificates on the Internet, reported at 130–150 million[11] ($\approx 2^{27}$).

Number of sessions $\#S \approx q_\mathrm{S} \in \{2^{35}, 2^{45}, 2^{55}\}$. Chromefirstofone[12] and Firefoxfirstofone[13] report that 76–98% of all web page accesses through these browsers are encrypted, with an active daily base of about 2 billion ($\approx 2^{30}$) users.We consider adversaries may easily see 2^{35} sessions and a global-scale attacker may have access to 2^{55} sessions over an extended timespan. Note that the number of send queries essentially corresponds to the number of sessions.

Number of RO queries $\#RO = q_\mathrm{RO} = \frac{t}{2^{10}}$. We fix this bound at a 2^{10}-fraction of the overall runtime accounting for all adversarial steps.

Diffie–Hellman groups and group order p. We consider all five elliptic curves standardized for firstofoneuse with TLS 1.3 (bit-security firstofonelevel $b$firstoftwo and group, order p in parentheses): `secp256r1` ($b = 128$, $p \approx 2^{256}$), `secp384r1` ($b = 192$, $p \approx 2^{384}$), `secp521r1` ($b = 256$, $p \approx 2^{521}$), `x25519` ($b = 128$, $p \approx 2^{252}$), and `x448` ($b = 224$, $p \approx 2^{446}$). We focus on elliptic curve groupsfirstofone only, as they provide high efficiency and the best known algorithms for solving discrete-log and DH problems are generic, allowing us to apply GGM bounds for firstoftwothe involved DDH and strong DH assumptionsDDH and strong DH.

Signature schemes. In order to unify the underlying hardness assumptions, we consider the ECDSA/EdDSA signature schemes standardized for use with TLS 1.3, based on the five elliptic curves above, treating their single-user unforgeability as equally hard as the corresponding discrete logarithmfirstofone problem.

Symmetric schemes and key/output/nonce lengths kl, ol, nl. Since our focus is mostly on evaluating ECDH parameters, we idealize the symmetric primitives (PRF, MAC, and hash function) in the random oracle model. Applying lengths standardized for TLS 1.3, we set the key and output length to $kl = ol = 256$ bits for 128-bit security curves and 384 bits for higher-security curves, corresponding to ciphersuites using SHA-256 or SHA-384. The nonce length is fixed to $nl = 256$ bits, again as in TLS 1.3.

[11] https://letsencrypt.org/stats/, https://trends.builtwith.com/ssl/traffic/Entire-Internet

[12] https://transparencyreport.google.com/https/

[13] https://telemetry.mozilla.org/

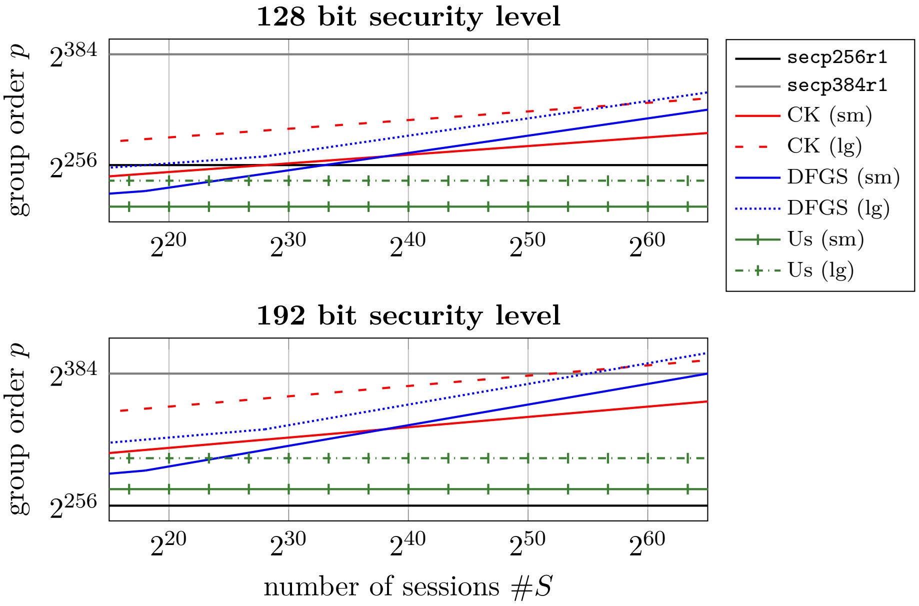


Figure 16: Elliptic curve group order (y axis) required to achieve 128-bit (top) and 192-bit (bottom) AKE security for SIGMA and TLS 1.3 based on the CK [?] SIGMA, DFGS [?] TLS 1.3, and our bounds (ours giving the same result for SIGMA and TLS 1.3), for a varying number of sessions $\#S$ (x axis). Both axes are in log-scale.
For each security and bound, we plot a smaller-resource "(sm)" setting with runtime $t = 2^{60}$, number of users $\#U = 2^{20}$, and number of random oracle queries $\#RO = 2^{50}$ and a larger-resource "(lg)" setting with $t = 2^{80}$, $\#U = 2^{30}$, and $\#RO = 2^{70}$. We let symmetric key/output lengths be 256 bits for 128-bit security and 384-bits for 192-bit security; nonce length is 256 bits. The group orders of NIST elliptic curves `secp256r1` ($p \approx 2^{256}$) and `secp384r1` ($p \approx 2^{384}$) are shown as horizontal lines for context.

Reveal and Test queries q_{RS}, q_{RL}, q_{T}. Using a generic reduction to single-user signature unforgeability, the number of RevLongTermKey, RevSessionKey, and Test queries do not affect the bounds; we hence do not place any constraints on them.

Fully-quantitative CK/DFGS bounds for SIGMA/TLS 1.3. For our evaluation, we need to reconstruct fully-quantitative security bounds from the more abstract prior security proofs for SIGMA by Canetti-Krawczyk [?] and for TLS 1.3 by Dowling et al. [?]. We report them in Appendix 10 for reference. In terms of their reduction to underlying DH problems, the CK SIGMA bound reduces to the DDH problem with a loss of $\#U \cdot \#S$, whereas the DFGS TLS 1.3 bound reduces to the strong DH problem with a loss of $(\#S)^2$.

Numerical advantage bounds for CK, DFGS, and ours. We report the numerical advantage bounds for SIGMA and TLS 1.3 based on prior (CK [?], DFGS [?]) and our bounds when ranging over the full parameter space detailed above in Table 2. Table 1 summarizes the key data points for 128-bit and 192-bit security levels.

Throughout Table 2, we assume that an adversary with running time t makes no more than $t \cdot 2^{-10}$ queries to its random oracles. We target the bit-security of whatever curve we use; this means that for b bits of security we want an advantage of $t/2^b$. If a bound does not achieve this

	Adv. resources					SIGMA		TLS 1.3	
t	$\#U$	$\#S$	$\#RO$	Curve (firstoftwobit security b, group order $p$bit sec. b, order p)	Target $t/2^b$	CK [?]	Us (Thm. 6.1)	DFGS [?]	Us (Thm. 8.1)
2^{40}	2^{20}	2^{35}	2^{30}	secp256r1 ($b=128, p\approx 2^{256}$)	2^{-88}	$\approx 2^{-101}$	$\approx 2^{-156}$	$\approx 2^{-104}$	$\approx 2^{-156}$
2^{40}	2^{20}	2^{45}	2^{30}	secp256r1 ($b=128, p\approx 2^{256}$)	2^{-88}	$\approx 2^{-91}$	$\approx 2^{-156}$	$\approx 2^{-84}$	$\approx 2^{-156}$
2^{40}	2^{20}	2^{55}	2^{30}	secp256r1 ($b=128, p\approx 2^{256}$)	2^{-88}	$\approx 2^{-81}$	$\approx 2^{-156}$	$\approx 2^{-64}$	$\approx 2^{-156}$
2^{40}	2^{30}	2^{35}	2^{30}	secp256r1 ($b=128, p\approx 2^{256}$)	2^{-88}	$\approx 2^{-81}$	$\approx 2^{-146}$	$\approx 2^{-104}$	$\approx 2^{-146}$
2^{40}	2^{30}	2^{45}	2^{30}	secp256r1 ($b=128, p\approx 2^{256}$)	2^{-88}	$\approx 2^{-71}$	$\approx 2^{-146}$	$\approx 2^{-84}$	$\approx 2^{-146}$
2^{40}	2^{30}	2^{55}	2^{30}	secp256r1 ($b=128, p\approx 2^{256}$)	2^{-88}	$\approx 2^{-61}$	$\approx 2^{-146}$	$\approx 2^{-64}$	$\approx 2^{-146}$
2^{40}	2^{20}	2^{35}	2^{30}	secp384r1 ($b=192, p\approx 2^{384}$)	2^{-152}	$\approx 2^{-229}$	$\approx 2^{-284}$	$\approx 2^{-232}$	$\approx 2^{-284}$
2^{40}	2^{20}	2^{45}	2^{30}	secp384r1 ($b=192, p\approx 2^{384}$)	2^{-152}	$\approx 2^{-219}$	$\approx 2^{-284}$	$\approx 2^{-212}$	$\approx 2^{-284}$
2^{40}	2^{20}	2^{55}	2^{30}	secp384r1 ($b=192, p\approx 2^{384}$)	2^{-152}	$\approx 2^{-209}$	$\approx 2^{-284}$	$\approx 2^{-192}$	$\approx 2^{-284}$
2^{40}	2^{30}	2^{35}	2^{30}	secp384r1 ($b=192, p\approx 2^{384}$)	2^{-152}	$\approx 2^{-209}$	$\approx 2^{-274}$	$\approx 2^{-232}$	$\approx 2^{-274}$
2^{40}	2^{30}	2^{45}	2^{30}	secp384r1 ($b=192, p\approx 2^{384}$)	2^{-152}	$\approx 2^{-199}$	$\approx 2^{-274}$	$\approx 2^{-212}$	$\approx 2^{-274}$
2^{40}	2^{30}	2^{55}	2^{30}	secp384r1 ($b=192, p\approx 2^{384}$)	2^{-152}	$\approx 2^{-189}$	$\approx 2^{-274}$	$\approx 2^{-192}$	$\approx 2^{-274}$
2^{40}	2^{20}	2^{35}	2^{30}	secp521r1 ($b=256, p\approx 2^{521}$)	2^{-216}	$\approx 2^{-298}$	$\approx 2^{-318}$	$\approx 2^{-282}$	$\approx 2^{-317}$
2^{40}	2^{20}	2^{45}	2^{30}	secp521r1 ($b=256, p\approx 2^{521}$)	2^{-216}	$\approx 2^{-288}$	$\approx 2^{-308}$	$\approx 2^{-262}$	$\approx 2^{-307}$
2^{40}	2^{20}	2^{55}	2^{30}	secp521r1 ($b=256, p\approx 2^{521}$)	2^{-216}	$\approx 2^{-278}$	$\approx 2^{-298}$	$\approx 2^{-242}$	$\approx 2^{-297}$
2^{40}	2^{30}	2^{35}	2^{30}	secp521r1 ($b=256, p\approx 2^{521}$)	2^{-216}	$\approx 2^{-288}$	$\approx 2^{-318}$	$\approx 2^{-282}$	$\approx 2^{-317}$
2^{40}	2^{30}	2^{45}	2^{30}	secp521r1 ($b=256, p\approx 2^{521}$)	2^{-216}	$\approx 2^{-278}$	$\approx 2^{-308}$	$\approx 2^{-262}$	$\approx 2^{-307}$
2^{40}	2^{30}	2^{55}	2^{30}	secp521r1 ($b=256, p\approx 2^{521}$)	2^{-216}	$\approx 2^{-268}$	$\approx 2^{-298}$	$\approx 2^{-242}$	$\approx 2^{-297}$
2^{40}	2^{20}	2^{35}	2^{30}	x25519 ($b=128, p\approx 2^{252}$)	2^{-88}	$\approx 2^{-97}$	$\approx 2^{-152}$	$\approx 2^{-100}$	$\approx 2^{-152}$
2^{40}	2^{20}	2^{45}	2^{30}	x25519 ($b=128, p\approx 2^{252}$)	2^{-88}	$\approx 2^{-87}$	$\approx 2^{-152}$	$\approx 2^{-80}$	$\approx 2^{-152}$
2^{40}	2^{20}	2^{55}	2^{30}	x25519 ($b=128, p\approx 2^{252}$)	2^{-88}	$\approx 2^{-77}$	$\approx 2^{-152}$	$\approx 2^{-60}$	$\approx 2^{-152}$
2^{40}	2^{30}	2^{35}	2^{30}	x25519 ($b=128, p\approx 2^{252}$)	2^{-88}	$\approx 2^{-77}$	$\approx 2^{-142}$	$\approx 2^{-100}$	$\approx 2^{-142}$
2^{40}	2^{30}	2^{45}	2^{30}	x25519 ($b=128, p\approx 2^{252}$)	2^{-88}	$\approx 2^{-67}$	$\approx 2^{-142}$	$\approx 2^{-80}$	$\approx 2^{-142}$
2^{40}	2^{30}	2^{55}	2^{30}	x25519 ($b=128, p\approx 2^{252}$)	2^{-88}	$\approx 2^{-57}$	$\approx 2^{-142}$	$\approx 2^{-60}$	$\approx 2^{-142}$
2^{40}	2^{20}	2^{35}	2^{30}	x448 ($b=224, p\approx 2^{446}$)	2^{-184}	$\approx 2^{-291}$	$\approx 2^{-318}$	$\approx 2^{-282}$	$\approx 2^{-317}$
2^{40}	2^{20}	2^{45}	2^{30}	x448 ($b=224, p\approx 2^{446}$)	2^{-184}	$\approx 2^{-281}$	$\approx 2^{-308}$	$\approx 2^{-262}$	$\approx 2^{-307}$
2^{40}	2^{20}	2^{55}	2^{30}	x448 ($b=224, p\approx 2^{446}$)	2^{-184}	$\approx 2^{-271}$	$\approx 2^{-298}$	$\approx 2^{-242}$	$\approx 2^{-297}$
2^{40}	2^{30}	2^{35}	2^{30}	x448 ($b=224, p\approx 2^{446}$)	2^{-184}	$\approx 2^{-271}$	$\approx 2^{-318}$	$\approx 2^{-282}$	$\approx 2^{-317}$
2^{40}	2^{30}	2^{45}	2^{30}	x448 ($b=224, p\approx 2^{446}$)	2^{-184}	$\approx 2^{-261}$	$\approx 2^{-308}$	$\approx 2^{-262}$	$\approx 2^{-307}$
2^{40}	2^{30}	2^{55}	2^{30}	x448 ($b=224, p\approx 2^{446}$)	2^{-184}	$\approx 2^{-251}$	$\approx 2^{-298}$	$\approx 2^{-242}$	$\approx 2^{-297}$
2^{60}	2^{20}	2^{35}	2^{50}	secp256r1 ($b=128, p\approx 2^{256}$)	2^{-68}	$\approx 2^{-61}$	$\approx 2^{-116}$	$\approx 2^{-64}$	$\approx 2^{-116}$
2^{60}	2^{20}	2^{45}	2^{50}	secp256r1 ($b=128, p\approx 2^{256}$)	2^{-68}	$\approx 2^{-51}$	$\approx 2^{-116}$	$\approx 2^{-44}$	$\approx 2^{-116}$
2^{60}	2^{20}	2^{55}	2^{50}	secp256r1 ($b=128, p\approx 2^{256}$)	2^{-68}	$\approx 2^{-41}$	$\approx 2^{-116}$	$\approx 2^{-24}$	$\approx 2^{-116}$
2^{60}	2^{30}	2^{35}	2^{50}	secp256r1 ($b=128, p\approx 2^{256}$)	2^{-68}	$\approx 2^{-41}$	$\approx 2^{-106}$	$\approx 2^{-64}$	$\approx 2^{-106}$
2^{60}	2^{30}	2^{45}	2^{50}	secp256r1 ($b=128, p\approx 2^{256}$)	2^{-68}	$\approx 2^{-31}$	$\approx 2^{-106}$	$\approx 2^{-44}$	$\approx 2^{-106}$
2^{60}	2^{30}	2^{55}	2^{50}	secp256r1 ($b=128, p\approx 2^{256}$)	2^{-68}	$\approx 2^{-21}$	$\approx 2^{-106}$	$\approx 2^{-24}$	$\approx 2^{-106}$
2^{60}	2^{20}	2^{35}	2^{50}	secp384r1 ($b=192, p\approx 2^{384}$)	2^{-132}	$\approx 2^{-189}$	$\approx 2^{-244}$	$\approx 2^{-192}$	$\approx 2^{-244}$
2^{60}	2^{20}	2^{45}	2^{50}	secp384r1 ($b=192, p\approx 2^{384}$)	2^{-132}	$\approx 2^{-179}$	$\approx 2^{-244}$	$\approx 2^{-172}$	$\approx 2^{-244}$
2^{60}	2^{20}	2^{55}	2^{50}	secp384r1 ($b=192, p\approx 2^{384}$)	2^{-132}	$\approx 2^{-169}$	$\approx 2^{-244}$	$\approx 2^{-152}$	$\approx 2^{-244}$
2^{60}	2^{30}	2^{35}	2^{50}	secp384r1 ($b=192, p\approx 2^{384}$)	2^{-132}	$\approx 2^{-169}$	$\approx 2^{-234}$	$\approx 2^{-192}$	$\approx 2^{-234}$
2^{60}	2^{30}	2^{45}	2^{50}	secp384r1 ($b=192, p\approx 2^{384}$)	2^{-132}	$\approx 2^{-159}$	$\approx 2^{-234}$	$\approx 2^{-172}$	$\approx 2^{-234}$
2^{60}	2^{30}	2^{55}	2^{50}	secp384r1 ($b=192, p\approx 2^{384}$)	2^{-132}	$\approx 2^{-149}$	$\approx 2^{-234}$	$\approx 2^{-152}$	$\approx 2^{-234}$
2^{60}	2^{20}	2^{35}	2^{50}	secp521r1 ($b=256, p\approx 2^{521}$)	2^{-196}	$\approx 2^{-278}$	$\approx 2^{-298}$	$\approx 2^{-250}$	$\approx 2^{-285}$
2^{60}	2^{20}	2^{45}	2^{50}	secp521r1 ($b=256, p\approx 2^{521}$)	2^{-196}	$\approx 2^{-268}$	$\approx 2^{-288}$	$\approx 2^{-240}$	$\approx 2^{-285}$
2^{60}	2^{20}	2^{55}	2^{50}	secp521r1 ($b=256, p\approx 2^{521}$)	2^{-196}	$\approx 2^{-258}$	$\approx 2^{-278}$	$\approx 2^{-222}$	$\approx 2^{-277}$
2^{60}	2^{30}	2^{35}	2^{50}	secp521r1 ($b=256, p\approx 2^{521}$)	2^{-196}	$\approx 2^{-268}$	$\approx 2^{-298}$	$\approx 2^{-250}$	$\approx 2^{-285}$
2^{60}	2^{30}	2^{45}	2^{50}	secp521r1 ($b=256, p\approx 2^{521}$)	2^{-196}	$\approx 2^{-258}$	$\approx 2^{-288}$	$\approx 2^{-240}$	$\approx 2^{-285}$
2^{60}	2^{30}	2^{55}	2^{50}	secp521r1 ($b=256, p\approx 2^{521}$)	2^{-196}	$\approx 2^{-248}$	$\approx 2^{-278}$	$\approx 2^{-222}$	$\approx 2^{-277}$
2^{60}	2^{20}	2^{35}	2^{50}	x25519 ($b=128, p\approx 2^{252}$)	2^{-68}	$\approx 2^{-57}$	$\approx 2^{-112}$	$\approx 2^{-60}$	$\approx 2^{-112}$
2^{60}	2^{20}	2^{45}	2^{50}	x25519 ($b=128, p\approx 2^{252}$)	2^{-68}	$\approx 2^{-47}$	$\approx 2^{-112}$	$\approx 2^{-40}$	$\approx 2^{-112}$
2^{60}	2^{20}	2^{55}	2^{50}	x25519 ($b=128, p\approx 2^{252}$)	2^{-68}	$\approx 2^{-37}$	$\approx 2^{-112}$	$\approx 2^{-20}$	$\approx 2^{-112}$
2^{60}	2^{30}	2^{35}	2^{50}	x25519 ($b=128, p\approx 2^{252}$)	2^{-68}	$\approx 2^{-37}$	$\approx 2^{-102}$	$\approx 2^{-60}$	$\approx 2^{-102}$
2^{60}	2^{30}	2^{45}	2^{50}	x25519 ($b=128, p\approx 2^{252}$)	2^{-68}	$\approx 2^{-27}$	$\approx 2^{-102}$	$\approx 2^{-40}$	$\approx 2^{-102}$
2^{60}	2^{30}	2^{55}	2^{50}	x25519 ($b=128, p\approx 2^{252}$)	2^{-68}	$\approx 2^{-17}$	$\approx 2^{-102}$	$\approx 2^{-20}$	$\approx 2^{-102}$
2^{60}	2^{20}	2^{35}	2^{50}	x448 ($b=224, p\approx 2^{446}$)	2^{-164}	$\approx 2^{-251}$	$\approx 2^{-298}$	$\approx 2^{-250}$	$\approx 2^{-285}$
2^{60}	2^{20}	2^{45}	2^{50}	x448 ($b=224, p\approx 2^{446}$)	2^{-164}	$\approx 2^{-241}$	$\approx 2^{-288}$	$\approx 2^{-234}$	$\approx 2^{-285}$
2^{60}	2^{20}	2^{55}	2^{50}	x448 ($b=224, p\approx 2^{446}$)	2^{-164}	$\approx 2^{-231}$	$\approx 2^{-278}$	$\approx 2^{-214}$	$\approx 2^{-277}$
2^{60}	2^{30}	2^{35}	2^{50}	x448 ($b=224, p\approx 2^{446}$)	2^{-164}	$\approx 2^{-231}$	$\approx 2^{-296}$	$\approx 2^{-250}$	$\approx 2^{-285}$
2^{60}	2^{30}	2^{45}	2^{50}	x448 ($b=224, p\approx 2^{446}$)	2^{-164}	$\approx 2^{-221}$	$\approx 2^{-288}$	$\approx 2^{-234}$	$\approx 2^{-285}$
2^{60}	2^{30}	2^{55}	2^{50}	x448 ($b=224, p\approx 2^{446}$)	2^{-164}	$\approx 2^{-211}$	$\approx 2^{-278}$	$\approx 2^{-214}$	$\approx 2^{-277}$
2^{80}	2^{20}	2^{35}	2^{70}	secp256r1 ($b=128, p\approx 2^{256}$)	2^{-48}	$\approx 2^{-21}$	$\approx 2^{-76}$	$\approx 2^{-24}$	$\approx 2^{-76}$
2^{80}	2^{20}	2^{45}	2^{70}	secp256r1 ($b=128, p\approx 2^{256}$)	2^{-48}	$\approx 2^{-11}$	$\approx 2^{-76}$	$\approx 2^{-4}$	$\approx 2^{-76}$
2^{80}	2^{20}	2^{55}	2^{70}	secp256r1 ($b=128, p\approx 2^{256}$)	2^{-48}	$\approx 2^{-1}$	$\approx 2^{-76}$	1	$\approx 2^{-76}$
2^{80}	2^{30}	2^{35}	2^{70}	secp256r1 ($b=128, p\approx 2^{256}$)	2^{-48}	$\approx 2^{-1}$	$\approx 2^{-66}$	$\approx 2^{-24}$	$\approx 2^{-66}$
2^{80}	2^{30}	2^{45}	2^{70}	secp256r1 ($b=128, p\approx 2^{256}$)	2^{-48}	1	$\approx 2^{-66}$	$\approx 2^{-4}$	$\approx 2^{-66}$
2^{80}	2^{30}	2^{55}	2^{70}	secp256r1 ($b=128, p\approx 2^{256}$)	2^{-48}	1	$\approx 2^{-66}$	1	$\approx 2^{-66}$
2^{80}	2^{20}	2^{35}	2^{70}	secp384r1 ($b=192, p\approx 2^{384}$)	2^{-112}	$\approx 2^{-149}$	$\approx 2^{-204}$	$\approx 2^{-152}$	$\approx 2^{-204}$
2^{80}	2^{20}	2^{45}	2^{70}	secp384r1 ($b=192, p\approx 2^{384}$)	2^{-112}	$\approx 2^{-139}$	$\approx 2^{-204}$	$\approx 2^{-132}$	$\approx 2^{-204}$
2^{80}	2^{20}	2^{55}	2^{70}	secp384r1 ($b=192, p\approx 2^{384}$)	2^{-112}	$\approx 2^{-129}$	$\approx 2^{-204}$	$\approx 2^{-112}$	$\approx 2^{-204}$
2^{80}	2^{30}	2^{35}	2^{70}	secp384r1 ($b=192, p\approx 2^{384}$)	2^{-112}	$\approx 2^{-129}$	$\approx 2^{-194}$	$\approx 2^{-152}$	$\approx 2^{-194}$
2^{80}	2^{30}	2^{45}	2^{70}	secp384r1 ($b=192, p\approx 2^{384}$)	2^{-112}	$\approx 2^{-119}$	$\approx 2^{-194}$	$\approx 2^{-132}$	$\approx 2^{-194}$
2^{80}	2^{30}	2^{55}	2^{70}	secp384r1 ($b=192, p\approx 2^{384}$)	2^{-112}	$\approx 2^{-109}$	$\approx 2^{-194}$	$\approx 2^{-112}$	$\approx 2^{-194}$
2^{80}	2^{20}	2^{35}	2^{70}	secp521r1 ($b=256, p\approx 2^{521}$)	2^{-176}	$\approx 2^{-258}$	$\approx 2^{-278}$	$\approx 2^{-210}$	$\approx 2^{-245}$
2^{80}	2^{20}	2^{45}	2^{70}	secp521r1 ($b=256, p\approx 2^{521}$)	2^{-176}	$\approx 2^{-248}$	$\approx 2^{-268}$	$\approx 2^{-200}$	$\approx 2^{-245}$
2^{80}	2^{20}	2^{55}	2^{70}	secp521r1 ($b=256, p\approx 2^{521}$)	2^{-176}	$\approx 2^{-238}$	$\approx 2^{-258}$	$\approx 2^{-190}$	$\approx 2^{-245}$
2^{80}	2^{30}	2^{35}	2^{70}	secp521r1 ($b=256, p\approx 2^{521}$)	2^{-176}	$\approx 2^{-248}$	$\approx 2^{-278}$	$\approx 2^{-210}$	$\approx 2^{-245}$
2^{80}	2^{30}	2^{45}	2^{70}	secp521r1 ($b=256, p\approx 2^{521}$)	2^{-176}	$\approx 2^{-238}$	$\approx 2^{-268}$	$\approx 2^{-200}$	$\approx 2^{-245}$
2^{80}	2^{30}	2^{55}	2^{70}	secp521r1 ($b=256, p\approx 2^{521}$)	2^{-176}	$\approx 2^{-228}$	$\approx 2^{-258}$	$\approx 2^{-190}$	$\approx 2^{-245}$
2^{80}	2^{20}	2^{35}	2^{70}	x25519 ($b=128, p\approx 2^{252}$)	2^{-48}	$\approx 2^{-17}$	$\approx 2^{-72}$	$\approx 2^{-20}$	$\approx 2^{-72}$
2^{80}	2^{20}	2^{45}	2^{70}	x25519 ($b=128, p\approx 2^{252}$)	2^{-48}	$\approx 2^{-7}$	$\approx 2^{-72}$	1	$\approx 2^{-72}$
2^{80}	2^{20}	2^{55}	2^{70}	x25519 ($b=128, p\approx 2^{252}$)	2^{-48}	1	$\approx 2^{-72}$	1	$\approx 2^{-72}$
2^{80}	2^{30}	2^{35}	2^{70}	x25519 ($b=128, p\approx 2^{252}$)	2^{-48}	1	$\approx 2^{-62}$	$\approx 2^{-20}$	$\approx 2^{-62}$
2^{80}	2^{30}	2^{45}	2^{70}	x25519 ($b=128, p\approx 2^{252}$)	2^{-48}	1	$\approx 2^{-62}$	1	$\approx 2^{-62}$
2^{80}	2^{30}	2^{55}	2^{70}	x25519 ($b=128, p\approx 2^{252}$)	2^{-48}	1	$\approx 2^{-62}$	1	$\approx 2^{-62}$
2^{80}	2^{20}	2^{35}	2^{70}	x448 ($b=224, p\approx 2^{446}$)	2^{-144}	$\approx 2^{-211}$	$\approx 2^{-266}$	$\approx 2^{-210}$	$\approx 2^{-245}$
2^{80}	2^{20}	2^{45}	2^{70}	x448 ($b=224, p\approx 2^{446}$)	2^{-144}	$\approx 2^{-201}$	$\approx 2^{-266}$	$\approx 2^{-194}$	$\approx 2^{-245}$
2^{80}	2^{20}	2^{55}	2^{70}	x448 ($b=224, p\approx 2^{446}$)	2^{-144}	$\approx 2^{-191}$	$\approx 2^{-258}$	$\approx 2^{-174}$	$\approx 2^{-245}$
2^{80}	2^{30}	2^{35}	2^{70}	x448 ($b=224, p\approx 2^{446}$)	2^{-144}	$\approx 2^{-191}$	$\approx 2^{-256}$	$\approx 2^{-210}$	$\approx 2^{-245}$
2^{80}	2^{30}	2^{45}	2^{70}	x448 ($b=224, p\approx 2^{446}$)	2^{-144}	$\approx 2^{-181}$	$\approx 2^{-256}$	$\approx 2^{-194}$	$\approx 2^{-245}$
2^{80}	2^{30}	2^{55}	2^{70}	x448 ($b=224, p\approx 2^{446}$)	2^{-144}	$\approx 2^{-171}$	$\approx 2^{-256}$	$\approx 2^{-174}$	$\approx 2^{-245}$

Table 2: firstoftwoAdvantages of a key exchange adversary with given resources t (running time), $\#U$ (number of users), $\#S$ (number of sessions), and $\#RO$ (number of random oracle queries), in breaking the security of the SIGMA and TLS 1.3 protocols when instantiated with the given curves (bit security b and group order p in parentheses), based on the prior bounds by Canetti-Krawczyk [?] resp. Dowling et al. [?], and the bounds we establish (Theorem 6.1 and 8.1). Target indicates the maximal advantage $t/2^b$ tolerable when aiming for the respective curve's security level b; entries in red-shaded cells miss that target. See Section 9 for further details. Advantages of a key exchange adversary with given resources in breaking the security of the SIGMA and TLS 1.3 protocols. See Section 9 for further details.

goal, we color it red. We consider runtimes between 2^{40} and 2^{80}, a total number of users between to vary between 2^{20} and 2^{30}, and a total number of sessions between 2^{35} and 2^{55} (see above for the discussion of these parameter choices). We evaluate these parameters in relation to all of the elliptic curve groups standardized for use with TLS 1.3. We idealize symmetric primitives, assuming the use of 256-bit keys in conjunction with 128-bit security curves and 384-bit keys in conjunction with higher-security curves, this corresponds to the available SHA-256 and SHA-384 functions in TLS 1.3. The nonce length is fixed to 256 bits (as in TLS 1.3).

Our bounds do better than the CK [?] and DFGS [?] bounds across all considered parameters and always meet the security targets, which these prior bounds fail to meet for `secp256r1` and `x25519` for almost all parameters, but notably also for the 192-bit security level of curve `secp384r1` for large-scale parameters. We improve over prior bounds by at least 20 and up to 85 bits of security for SIGMA, and by at least 35 and up to 92 bits of security for TLS 1.3.

In comparison, the TLS 1.3 bounds from the concurrent work by Diemert and Jager [?] yield bit security levels similar to ours for TLS 1.3: While some sub-terms in their bound are slightly worse (esp. for strong DH), the dominating sub-terms are the same.

Group size requirements based on CK, DFGS, and our bound. Finally, let us take a slightly different perspective on what the prior and our bounds tell us: Figure 9 illustrates the group size required to achieve 128-bit resp. 192-bit AKE security for SIGMA and TLS 1.3 based on the different bounds, dependent on a varying number of sessions $\#S$. The CK SIGMA and our SIGMA and TLS 1.3 bounds are dominated by the signature scheme advantage (with a $\#S \cdot (\#U)^2$ loss for CK and a $\#U$ loss for our bound); the DFGS TLS 1.3 bound instead is mostly dominated by the $(\#S)^2$–loss reduction to strong DH. The CK and DFGS bounds require the use of larger, less efficient curves to achieve 128-bit security even for 2^{35} sessions. For large-scale attackers, they similarly require a larger curve than `secp384r1` above about 2^{55} sessions. We highlight that, in contrast, with our bounds a curve with 128-bit, resp. 192-bit, security is sufficient to guarantee the same security level for SIGMA and TLS 1.3, for both small- and large-scale adversaries and for very conservative bounds on the number of sessions.

10 Evaluation Details

10.1 Fully-quantitative CK SIGMA Bound

Recall our security bound for SIGMA/SIGMA-I from Theorem 6.1:

$$\begin{aligned}
&\mathsf{Adv}^{\mathsf{KE\text{-}SEC}}_{\mathrm{SIGMA\text{-}I}}(t, q_{\mathrm{N}}, q_{\mathrm{S}}, q_{\mathrm{RS}}, q_{\mathrm{RL}}, q_{\mathrm{T}}) \\
&\quad \leq \frac{3q_{\mathrm{S}}^2}{2^{nl+1} \cdot p} + \mathsf{Adv}^{\mathsf{stDH}}_{\mathbb{G}}(t_{\mathcal{B}_1}, q_{\mathrm{RO}}) + \mathsf{Adv}^{\mathsf{mu\text{-}PRF}}_{\mathsf{PRF}}(t_{\mathcal{B}_2}, q_{\mathrm{S}}, 3q_{\mathrm{S}}, 3) \\
&\qquad + \mathsf{Adv}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{S}}(t_{\mathcal{B}_3}, q_{\mathrm{N}}, q_{\mathrm{S}}, q_{\mathrm{S}}, q_{\mathrm{RL}}) + \mathsf{Adv}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{M}}(t_{\mathcal{B}_4}, q_{\mathrm{S}}, q_{\mathrm{S}}, 1, q_{\mathrm{S}}, 1, 0).
\end{aligned}$$

Comparing this boundto the original security proof for SIGMA by Canetti and Krawczyk [?] (CK) faces two complications. First, we must reconstruct a concrete security bound from the CK proof, which merely refers to the decisional Diffie–Hellman and "standard security notions" for digital signatures, MACs, and PRFs (i.e., single-user EUF-CMA and PRF security). Second, the CK result is given in a stronger security model for key exchange [?] which allows state-reveal attacks. Further, the CK proof assumes out-of-band unique session identifiers, whereas protocols in practice have to establish those from, e.g., nonces (introducing a corresponding collision bound as in our analysis). We are therefore inherently constrained to compare qualitatively different security properties here.

Let us informally consider a game-based definition of the CK model [?] in the same style as our AKE model (cf. Definition 2.1), capturing the same oracles plus an additional state-reveal oracle, with $q_{\mathrm{RS_T}}$ denoting the number of queries to this oracle, and session identifiers that, like ours, consist of the session and peers' nonces and DH shares. Translating the SIGMA-I security proof from [?, Theorem 6 in the full version], we obtained the following concrete security bound:

$$\begin{aligned}
&\mathsf{Adv}^{\mathrm{CK}}_{\mathrm{SIGMA\text{-}I}}(t, q_{\mathrm{N}}, q_{\mathrm{S}}, q_{\mathrm{RS}}, q_{\mathrm{RL}}, q_{\mathrm{RS_T}}, q_{\mathrm{T}})\\
&\quad\leq \frac{2q_{\mathrm{S}}^2}{2^{nl}\cdot p} + \mathsf{Adv}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{S}}(t_{\mathcal{B}_1}, q_{\mathrm{N}}, q_{\mathrm{S}}, q_{\mathrm{S}}, q_{\mathrm{RL}}) \qquad /\!\!/ \text{ sid collision \& property P1}\\
&\qquad + q_{\mathrm{N}}\cdot q_{\mathrm{S}}\cdot\Big(\mathsf{Adv}^{\mathsf{DDH}}_{\mathbb{G}}(t_{\mathcal{B}_2}) + \mathsf{Adv}^{\mathsf{mu\text{-}PRF}}_{\mathsf{PRF}}(t_{\mathcal{B}_5}, 1, 3) \qquad /\!\!/ \text{ property P2} \ldots\\
&\qquad + (q_{\mathrm{N}}+1)\cdot\mathsf{Adv}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{S}}(t_{\mathcal{B}_3}, 1, q_{\mathrm{S}}, q_{\mathrm{S}}, 0) + \mathsf{Adv}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{M}}(t_{\mathcal{B}_4}, 1, 2, 2, 2, 2, 0)\Big),
\end{aligned}$$

where nl is the nonce length, $\mathbb{G}$ the used Diffie–Hellman group of prime order p, the number of test queries is restricted to $q_{\mathrm{T}} = 1$, and $\mathcal{B}_i$ (for $i = 1, \ldots, 5$) are the described reductions for property P1 and P2 in [?, Theorem 6 in the full version] all running in time $t_{\mathcal{B}_i} \approx t$. For simplicity, we present the above bound in terms of "multi-user" PRF, signature, and MAC advantages for a single user $q_{\mathrm{N_W}} = 1$, which are equivalent to the corresponding single-user advantages (cf. Section 3).

10.2 Fully-quantitative DFGS TLS 1.3 Bound

Recall our security bound for TLS 1.3 from Theorem 8.1:

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{KE\text{-}SEC}}_{\mathrm{TLS\,1.3}}(t, q_{\mathrm{N}}, q_{\mathrm{S}}, q_{\mathrm{RS}}, q_{\mathrm{RL}}, q_{\mathrm{T}}) &\leq \frac{3q_{\mathrm{S}}^2}{2^{nl+1}\cdot p} + \mathsf{Adv}^{\mathsf{CR}}_{\mathsf{H}}(t_{\mathcal{B}_1})\\
&+ 2\cdot\mathsf{Adv}^{\mathsf{stDH}}_{\mathbb{G}}(t_{\mathcal{B}_2}, q_{\mathrm{RO}}) + \frac{q_{\mathrm{RO}}\cdot q_{\mathrm{S}}}{2^{kl-1}} + \mathsf{Adv}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{S}}(t_{\mathcal{B}_3}, q_{\mathrm{N}}, q_{\mathrm{S}}, q_{\mathrm{S}}, q_{\mathrm{RL}})\\
&+ \mathsf{Adv}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{HMAC}}(t_{\mathcal{B}_4}, q_{\mathrm{S}}, q_{\mathrm{S}}, 1, q_{\mathrm{S}}, 1, 0).
\end{aligned}$$

We compare this bound with the bound of Dowling et al. [?] (DFGS). Note that this bound is established in a multi-stage key exchange model [?], here we focus only on the main application key derivation, as in our proof. The DFGS bound needs instantiation through the random oracle only in one step (the PRF-ODH assumption on $\mathsf{HKDF.Extract}$) while other PRF steps remain in the standard model. Our proof instead models both $\mathsf{HKDF.Extract}$ and $\mathsf{HKDF.Expand}$ as random oracles. Translating the bound from [?, Theorems 5.1, 5.2] yields:

$$\begin{aligned}
&\mathsf{Adv}^{\mathrm{DFGS}}_{\mathrm{TLS\,1.3}}(t, q_{\mathrm{N}}, q_{\mathrm{S}}, q_{\mathrm{RS}}, q_{\mathrm{RL}}, q_{\mathrm{T}})\\
&\quad\leq \frac{q_{\mathrm{S}}^2}{2^{nl}\cdot p} + q_{\mathrm{S}}\cdot\Bigg(\mathsf{Adv}^{\mathsf{CR}}_{\mathsf{H}}(t_{\mathcal{B}_1}) + q_{\mathrm{N}}\cdot\mathsf{Adv}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{S}}(t_{\mathcal{B}_2}, 1, q_{\mathrm{S}}, q_{\mathrm{S}}, 0)\\
&\quad + q_{\mathrm{S}}\cdot\Big(\mathsf{Adv}^{\mathsf{dual\text{-}snPRF\text{-}ODH}}_{\mathsf{HKDF.Extract},\mathbb{G}}(t_{\mathcal{B}_3}) + \mathsf{Adv}^{\mathsf{mu\text{-}PRF}}_{\mathsf{HKDF.Expand}}(t_{\mathcal{B}_4}, 1, 3, 3, 0)\\
&\quad + 2\cdot\mathsf{Adv}^{\mathsf{mu\text{-}PRF}}_{\mathsf{HKDF.Expand}}(t_{\mathcal{B}_5}, 1, 2, 2, 0) + \mathsf{Adv}^{\mathsf{mu\text{-}PRF}}_{\mathsf{HKDF.Extract}}(t_{\mathcal{B}_6}, 1, 1, 1, 0)\\
&\quad + \mathsf{Adv}^{\mathsf{mu\text{-}PRF}}_{\mathsf{HKDF.Expand}}(t_{\mathcal{B}_7}, 1, 1, 1, 0)\Big)\Bigg).
\end{aligned}$$

Let us further unpack the PRF-ODH term. Following Brendel et al. [**?**], it can be reduced to the strong Diffie–Hellman assumption modeling $\mathsf{HKDF.Extract}$ as a random oracle.[14] In this reduction, the single DH oracle query is checked against each random oracle query via the strong-DH oracle, hence establishing the following bound:

$$\mathsf{Adv}^{\mathsf{dual\text{-}snPRF\text{-}ODH}}_{\mathsf{RO},\mathbb{G}}(t_{\mathcal{B}_3}, q_{\mathrm{RO}}) \leq \mathsf{Adv}^{\mathsf{stDH}}_{\mathbb{G}}(t_{\mathcal{B}_3}, q_{\mathrm{RO}}).$$

Acknowledgments

We thank Mihir Bellare for insightful discussions and helpful comments, and Denis Diemert and Tibor Jager for their kind handling of our concurrent work. We thank the anonymous reviewers for valuable comments. Both authors were supported in part by National Science Foundation (NSF) grant CNS-1717640. Felix Günther has been supported in part by Research Fellowship grant GU 1859/1-1 of the German Research Foundation (DFG).

[14] The same paper suggests that a standard-model instantiation of the PRF-ODH assumption via an algebraic black-box reduction to common cryptographic problems is implausible.