Tighter Proofs for the SIGMA and TLS 1.3 Key Exchange Protocols

Tighter Proofs for the SIGMA and TLS 1.3 Key Exchange Protocols

Hannah Davis[1(✉)] and Felix Günther[2(✉)]

Department of Computer Science & Engineering, UC San Diego, La Jolla, USA
Department of Computer Science, ETH Zürich, Zürich, Switzerland
h3davis@eng.ucsd.edu mail@felixguenther.info

Abstract. We give new, fully-quantitative and concrete bounds that justify the SIGMA and TLS 1.3 key exchange protocols not just in principle, but in practice. By this we mean that, for standardized elliptic curve group sizes, the overall protocol actually achieves the intended security level.
Prior work gave reductions of both protocols' security to the underlying building blocks that were loose (in the number of users and/or sessions), so loose that they gave no guarantees for practical parameters. Adapting techniques by Cohn-Gordon et al. (Crypto 2019), we give reductions for SIGMA and TLS 1.3 to the strong Diffie–Hellman problem which are tight, and prove that this problem is as hard as solving discrete logarithms in the generic group model. Leveraging our tighter bounds, we meet the protocols' targeted security levels when instantiated with standardized curves and improve over prior bounds by up to over 90 bits of security across a range of real-world parameters.

Keywords: Key exchange, SIGMA, TLS 1.3, security bounds, tightness

1 Introduction

The Transport Layer Security (TLS) protocol [41] is responsible for securing billions of Internet connections every day. Usage statistics for Google Chrome and Mozilla Firefox report that 76–98% of all web page accesses are encrypted.[1] 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 [32] for the Internet Key Exchange (IKE) protocol [28] of IPsec [31], 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

[1] https://transparencyreport.google.com/, https://telemetry.mozilla.org/

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.

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 [12,10,16]. 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

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

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 [17] and TLS 1.3 [22,35,26,24]) 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 practical parameters. While the asymptotic bound tells us that scaling up the parameters for X (say, the DDH problem [14]) 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`,

Adv. resources					SIGMA		TLS 1.3	
t	$\#U$	$\#S$	Curve	Target	CK [17]	Us (Thm. 4)	DFGS [24]	Us (Thm. 5)
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 [17] resp. Dowling et al. [24], and the bounds we establish (Theorem 4 and 5). 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 7 for full details and curves `secp521r1` and `x448`.

`secp384r1` [39], or curve `x25519` [37] and idealizing the protocols' other components (see Section 7 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 Internet, $\#S$ ranging in 2^{35}–2^{55} based on 2^{32} Internet users and 2^{33} daily Google searches[2]). In the security analysis by Canetti and Krawczyk [17] (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. [24] (DFGS) for TLS 1.3 reduces to the strong Diffie–Hellman problem [1]—via the PRF-ODH assumption [29,15]—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 [17] and DFGS [24] advantage bounds for SIGMA and TLS 1.3 with

[2] `https://letsencrypt.org/stats/`, `https://www.internetlivestats.com/`

curves `secp256r1` and `x25519` fall 6–11 bits below the target of 2^{-68} for 128-bit security.
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.

Contributions. Most prior results in tightly secure key exchange (e.g., [4,27]) 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. 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 [1], which in addition we prove is as hard as the discrete logarithm problem in the generic group model (GGM) [42,38]. Instantiating our bounds shows that, with standardized real-world parameters, we achieve the intended security levels even when considering powerful, globally-scaled attackers.

Tighter security proof of SIGMA(-I). We establish fully quantitative security bounds for SIGMA and its identity-protecting variant SIGMA-I [32] in Sections 3 and 4. Our result is for BR-like [12] key exchange security and gives a tight reduction to the strong Diffie–Hellman problem [1] 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. [19] in the random oracle model [11]. The latter mu-security bounds are essentially equivalent to the corresponding bounds by CK [17]. 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 DDH [14] used in [17]), we prove (in Appendix C) that the strong DH problem, like DDH, is as hard as solving discrete logarithms in the generic group model [42,38].

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 [41], in Sections 5 and 6. The main quantitative improvement in our reduction is again a tight reduction to the strong DH problem, whereas prior bounds by DFGS [24] incurred a quadratic loss to the PRF-ODH assumption [29,15], a loss

which translates directly to strong DH [15]. While TLS 1.3 roughly follows the SIGMA-I design, its cascading key schedule impedes the precise technique of Cohn-Gordon et al. [19] 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 7, 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 standardized for use in TLS 1.3 [41], a summary of which is displayed in Table 1. We report that our tighter proofs indeed materialize for a wide range of real-world resource parameters. The resulting attacker advantages meet the targeted security levels of all five curves. In comparison to the prior CK [17] SIGMA and DFGS [24] 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.

Concurrent work. In concurrent and independent work, Diemert and Jager (DJ) [21] 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. [19], 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 7. 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 [21] in turn study tight composition with the TLS record protocol.

The DJ analysis is carried out in the multi-stage key exchange model [25], 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. [21, Theorem 6]). We instead model the key schedule's inner HKDF [34] 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 Model 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 [12] 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. [19], we formalize our model in a real-or-random version (following Abdalla, Fouque, and Pointcheval [3] with added forward secrecy [2]) 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 [25]) treatment and refer to the concurrent work by Diemert and Jager [21] for tight results for TLS 1.3 in a MSKE model.

In contrast to the work by Cohn-Gordon et al. [19] and Diemert and Jager [21], our model additionally captures explicit authentication through the ExplicitAuth predicate in Figure 1, ensuring sessions with non-corrupted peer accept with an honest partner session. We and [21] 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" [17] particularly applies to the SIGMA protocol family [32] as well as TLS 1.3 [41].

Key exchange protocols. We begin by formalizing the syntax of key exchange protocols. A key exchange protocol KE consists of three algorithms (KGen, Activate, Run) and an associated key space 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; 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).
- $skey \in \mathsf{KE.KS}$. The derived session key (in$\mathsf{KE.KS}$), 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.

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 [13]. 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 ensures session identifiers are set in a sound manner (non-colliding, ensuring agreement on session keys). $\mathsf{ExplicitAuth}$ encodes explicit authentication, requiring that accepted sessions agree on the intended peer (if non-corrupted). Finally, to capture key secrecy, we have to restrict the adversary to testing only *fresh* (i.e., not trivially compromised) sessions in order to exclude trivial attacks; this is ensured through Fresh.

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

Initialize:

1 $\mathsf{time} \leftarrow 0$; $\mathsf{users} \leftarrow 0$
2 $b \xleftarrow{\$} \{0,1\}$

NewUser:

3 $\mathsf{users} \leftarrow \mathsf{users} + 1$
4 $(pk_{\mathsf{users}}, sk_{\mathsf{users}}) \xleftarrow{\$} \mathsf{KGen}()$
5 $\mathsf{revltk}_{\mathsf{users}} \leftarrow \infty$
6 $peerpk[\mathsf{users}] \leftarrow pk_{\mathsf{users}}$
7 return pk_{users}

Send(u, i, m):

8 if $\pi^i_u = \perp$ then
9 $(peerid, role) \leftarrow m$
10 $(\pi^i_u, m') \xleftarrow{\$} \mathsf{Activate}(u, sk_u, peerid, peerpk, role)$
11 $\pi^i_u.\mathsf{t_{acc}} \leftarrow 0$
12 else
13 $(\pi^i_u, m') \xleftarrow{\$} \mathsf{Run}(u, sk_u, \pi^i_u, peerpk, m)$
14 if $\pi^i_u.status = \mathsf{accepted}$ then
15 $\mathsf{time} \leftarrow \mathsf{time} + 1$
16 $\pi^i_u.\mathsf{t_{acc}} \leftarrow \mathsf{time}$
17 return m'

RevSessionKey(u, i):

18 if $\pi^i_u = \perp$ or $\pi^i_u.status \neq \mathsf{accepted}$ then
19 return $\perp$
20 $\pi^i_u.\mathsf{revealed} \leftarrow \mathsf{true}$
21 return $\pi^i_u.skey$

RevLongTermKey(u):

22 $\mathsf{time} \leftarrow \mathsf{time} + 1$
23 $\mathsf{revltk}_u \leftarrow \mathsf{time}$
24 return sk_u

Test(u, i):

25 if $\pi^i_u = \perp$ or $\pi^i_u.status \neq \mathsf{accepted}$ or $\pi^i_u.\mathsf{tested}$ then
26 return $\perp$
27 $\pi^i_u.\mathsf{tested} \leftarrow \mathsf{true}$
28 $T \leftarrow T \cup \{\pi^i_u\}$
29 $k_0 \leftarrow \pi^i_u.skey$
30 $k_1 \xleftarrow{\$} \mathsf{KE.KS}$
31 return k_b

Finalize(b'):

32 if $\neg$Sound then return 1
33 if $\neg$ExplicitAuth then return 1
34 if $\neg$Fresh then $b' \leftarrow 0$
35 return $[[b = b']]$

Sound:

1 if $\exists$ distinct π^i_u, π^j_v, π^k_w with $\pi^i_u.sid = \pi^j_v.sid = \pi^k_w.sid$ then // no triple sid match
2 return false
3 if $\exists \pi^i_u, \pi^j_v$ with
$\pi^i_u.status = \pi^j_v.status = \mathsf{accepted}$
and $\pi^i_u.sid = \pi^j_v.sid$
and $\pi^i_u.peerid = v$ and $\pi^j_v.peerid = u$
and $\pi^i_u.role \neq \pi^j_v.role$, but $\pi^i_u.skey \neq \pi^j_v.skey$ then // partnering implies same key
4 return false
5 return true

ExplicitAuth:

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

Fresh:

1 for each $\pi^i_u \in T$
2 if $\pi^i_u.\mathsf{revealed}$ then
3 return false // tested session may not be revealed
4 if $\exists \pi^j_v \neq \pi^i_u : \pi^j_v.sid = \pi^i_u.sid$ and $(\pi^j_v.\mathsf{tested}$ or $\pi^j_v.\mathsf{revealed})$ then
5 return false // tested session's partnered session may not be tested or revealed
6 if $\mathsf{revltk}_{\pi^i_u.peerid} < \pi^i_u.\mathsf{t_{acc}}$ then
7 return false // tested session's peer may not be corrupted prior to acceptance
8 return true

Fig. 1. Key exchange security game.

We call two distinct sessions π_u^i and π_v^j *partnered* if $\pi_u^i.sid = \pi_v^j.sid$. We refer to sessions generated by $\mathsf{Activate}$ (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 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 $(t, q_{\mathrm{N}}, q_{\mathrm{S}}, q_{\mathrm{RS}}, q_{\mathrm{RL}}, q_{\mathrm{T}})$-$\mathsf{KE\text{-}SEC}$-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*.*

Security properties. We capture regular *key secrecy* of the main session key through Test queries, incorporating *explicit authentication* as well as ("perfect") *forward secrecy* by allowing corruption as long as each tested sessions accepted prior to corrupting its intended peer. This strengthens our model compared to that of Cohn-Gordon et al. [19] which only captures implicit authentication and weak forward secrecy; while Diemert and Jager [21] additionally treat the security of intermediate and further keys beyond the main session key in a multi-stage approach [25], but without capturing explicit authentication. Like [19,21], our model captures *key-compromise impersonation*, but not *session-state or randomness reveals* [16,36] or *post-compromise security* [18].

Multi-User Security Advantages. Before we continue to our technical results, let us briefly introduce notation and discuss the multi-user security of the involved building blocks: PRFs, digital signatures, MAC schemes, and hash functions. We defer full definitions to Appendix B and only explain how to read the advantage bounds here.

- **PRF:** $\mathsf{Adv}^{\mathsf{mu\text{-}PRF}}_{\mathsf{PRF}}(t, q_{\mathrm{NW}}, q_{\mathrm{FN}}, q_{\mathrm{FN/U}})$**.** The maximal advantage in distinguishing PRF from a random function of any adversary running in time t with access to at most q_{NW} users, making at most q_{FN} function queries overall and $q_{\mathrm{FN/U}}$ function queries per user.
- **Signature:** $\mathsf{Adv}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{S}}(t, q_{\mathrm{NW}}, q_{\mathrm{SG}}, q_{\mathrm{SG/U}}, q_{\mathrm{C}})$**.** The maximal advantage for an existential signature forgery for S of any adversary running in time t with access to at most q_{NW} users, making at most $q_{\mathrm{SG}}/q_{\mathrm{SG/U}}$ signing queries total/per user, allowed to adaptively corrupt at most q_{C} users.
- **MAC:** $\mathsf{Adv}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{M}}(t, q_{\mathrm{NW}}, q_{\mathrm{TG}}, q_{\mathrm{TG/U}}, q_{\mathrm{VF}}, q_{\mathrm{VF/U}}, q_{\mathrm{C}})$**.** The maximal advantage for an existential MAC forgery for M of any adversary running in time t with access to at most q_{NW} users, making at most $q_{\mathrm{TG}}/q_{\mathrm{TG/U}}$ and $q_{\mathrm{VF}}/q_{\mathrm{VF/U}}$ tagging resp. verification queries total/per user, allowed to adaptively corrupt at most q_{C} users.

Hash: $\mathsf{Adv}^{\mathsf{CR}}_{\mathsf{H}}(t)$. The advantage of a given adversary running in time t in outputting a hash collision under H.

Strong Diffie–Hellman GGM bound. The strong Diffie–Hellman (strong DH) assumption, a weakening of the gap DH assumption [40], states that solving the computational DH problem given a restricted decisional DH [14] oracle is hard.

Definition 2 (Strong Diffie–Hellman problem [40]). *Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order q. Let $\mathsf{DDH}(X, Y, Z) := [[X^{\log_g(Y)} = Z]]$ be a decisional Diffie–Hellman oracle. We define*

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

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) DH assumption has been deployed in numerous works to analyze practical key exchange designs, directly or through the PRF-ODH assumption [29,15] it supports, including [29,25,22,35,23,26,24] as well as in the closely related works on practical tightness by Cohn-Gordon et al. [19] and Diemert and Jager [21]. To argue that it is reasonable to rely on the strong DH assumption, we turn to the generic group model [42,38]. 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 [42] 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, CDH, or DDH problem. We claim, and prove in Appendix C, 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. *Let $\mathbb{G}$ be a group with prime order p. In the generic group model, $\mathsf{Adv}^{\mathsf{stDH}}_{\mathbb{G}}(t, q) \leq 4t^2/p$.*

3 The SIGMA Protocol

The SIGMA family of key exchange protocols introduced by Krawczyk [32,33] describes several variants for building authenticated Diffie–Hellman key exchange using the "SIGn-and-MAc" approach. Its design has been adopted in several Internet security protocols, including, e.g., the Internet Key Exchange protocol [28,30] as part of the IPsec Internet security protocol [31] and the newest version 1.3 of the Transport Layer Security (TLS) protocol [41].

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

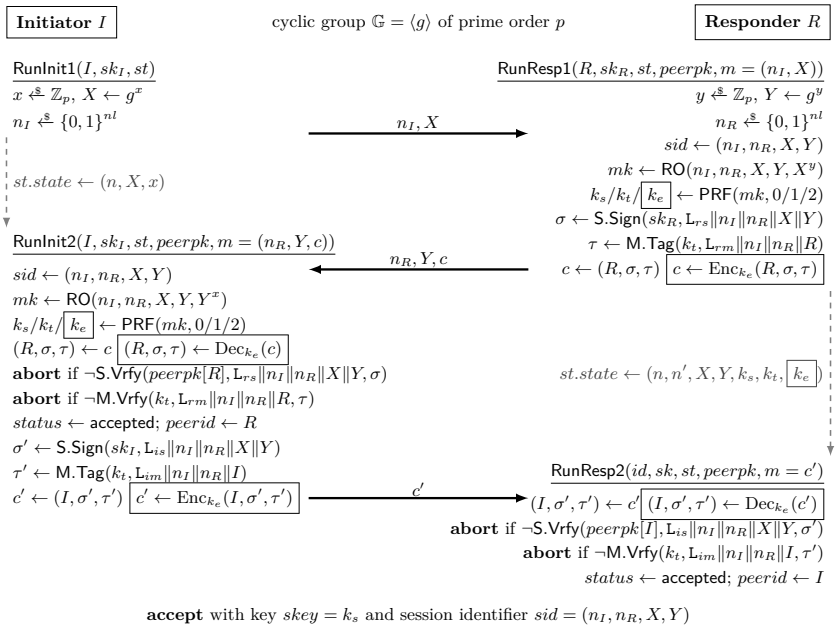

Fig. 2. The SIGMA/SIGMA-I protocol flow diagram. Boxed code is only performed in the SIGMA-I variant. Values $\mathtt{L}_x$ indicate label strings (distinct per x).

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 [33, Appendix B]. We illustrate these protocol flows in Figure 2.

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.

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 $sid = (n_I, n_R, X, Y)$.

4 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 4. *Let the SIGMA-I protocol be as specified in Figure 2 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 $(t, q_{\mathrm{N}}, q_{\mathrm{S}}, q_{\mathrm{RS}}, q_{\mathrm{RL}}, q_{\mathrm{T}})$-$\mathsf{KE}$-$\mathsf{SEC}$-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 $i = 2, \ldots, 4$) close to that of $\mathcal{A}$, such that*

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

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 [6], 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 outline. We defer the detailed game-based description of the proof to the full version [20] and only outline its core and novel technical steps here. We give a more detailed proof for our TLS 1.3 bound in Section 6 which requires careful handling of the more complex key schedule, but is still structurally close.

The heart of the proof is the reduction to the strong DH problem. In prior analyses of SIGMA and TLS 1.3, this reduction embeds a DH challenge into a single tested session. This technique incurs a loss in the number of sessions because the reduction must guess in advance which session will be tested. Translating techniques from Cohn-Gordon et al. [19], we instead use the random self-reducibility of DH to embed a single challenge into every session which could possibly accept and be tested without violating the Fresh predicate.

We can divide all sessions into two categories: (A) those who receive nonces or DH shares that have been tampered with by an adversary and (B) those who receive unaltered nonces and DH shares from an honest peer. Embedding a DH challenge into each of these types of sessions must be addressed differently.

If an adversary controls the DH share received by an honest session (category (A)), it can compute that session's DH secret, from which are derived master key, session key, and MAC key. If such a session has an embedded challenge, the simulator cannot honestly produce the proper master key. Instead, it uses the strong DH oracle to detect if the adversary ever makes an RO query containing the session's nonces, DH shares, and the corresponding DH secret, and it programs the response to this query to maintain consistency. The reduction also cannot produce the proper master key for sessions in category (B); however, it can again use the strong DH oracle to detect RO queries containing a valid DH secret that would output the proper master key. This secret can be used to extract the challenge secret and hence win the strong DH game. One particular nuance here is that checking each RO query for every session's DH secret would lead to a quadratic loss in the number of strong DH oracle queries. We maintain tightness by instead using the nonces and group elements in the RO query to identify the relevant sessions and efficiently program responses.

For sessions in category (B), the master key is now chosen uniformly at random. Invoking PRF security allows the session, traffic encryption, and MAC keys to be selected at random as well. Each accepting session must receive a valid signature and MAC tag on its nonces and group elements. Excluding the small probability that nonces and group elements collide between honest sessions, the adversary can only produce these by corrupting a long-term key or by forgery. The former approach violates the Fresh predicate; the latter violates the EUF-CMA security of either the signature or MAC scheme. Therefore, these sessions will accept only if they complete an entire protocol execution without tampering with an honest peer holding the same master key and thus same session key.

For sessions in category (A), the master key may be known to the adversary. However, these sessions still must receive a valid signature to accept. Since the nonces and group elements were tampered with, no honest session will produce this signature. Again, the adversary must resort to either corruption or forgery, hence violating either freshness or signature EUF-CMA security. □

5 The TLS 1.3 Handshake Protocol

The Transport Layer Security (TLS) protocol in version 1.3 [41] 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.[3] 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.

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.

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 keys and accordingly leave out some computations and additional messages. We illustrate the handshake protocol and its accompanying key schedule in Figure 3, the latter deriving keys in the extract-then-expand paradigm of the HKDF key derivation function [34].[4]

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 $\mathrm{DHE} = g^{xy}$ using $\mathsf{HKDF.Extract}$ with a constant salt input. 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 certificate, as well as a MAC over the (hashed) transcript up to incl. their signatures. Note the similarity to SIGMA-I here: each party signs both nonces

[3] 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.

[4] $\mathsf{HKDF.Extract}(XTS, SKM)$ on input salt XTS and source key material SKM outputs a pseudorandom key PRK. $\mathsf{HKDF.Expand}(PRK, CTXinfo)$ on input a pseudorandom key PRK and context information $CTXinfo$ outputs pseudorandom key material KM.

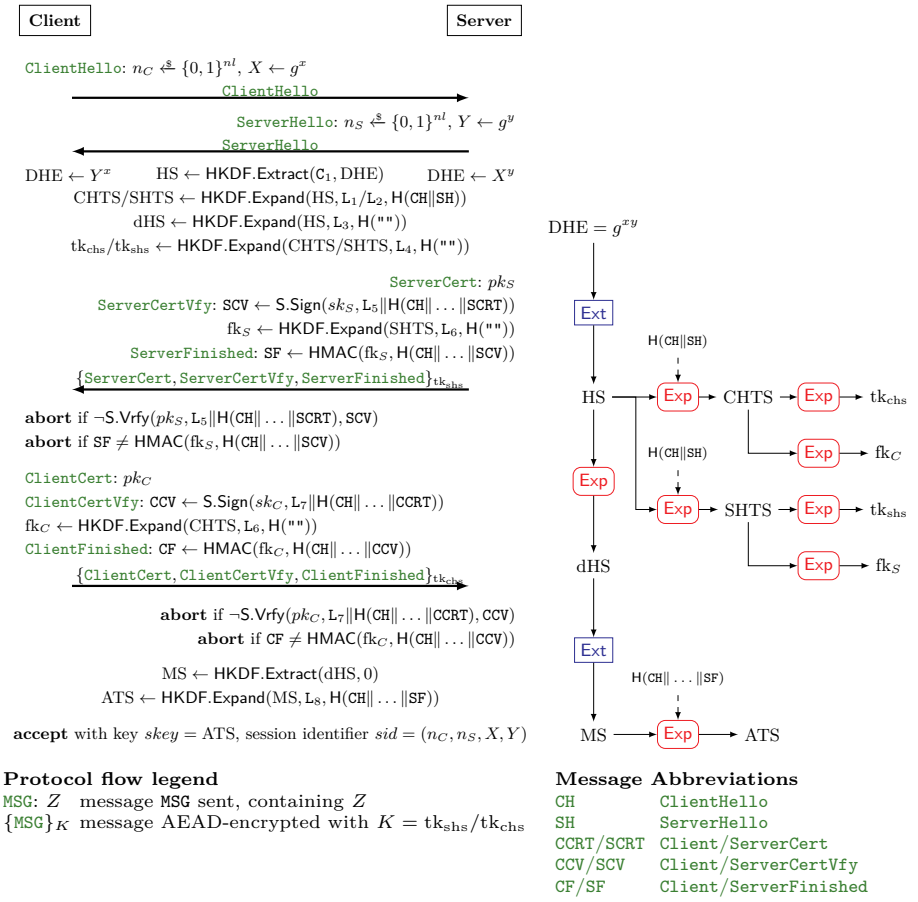

Fig. 3. The simplified TLS 1.3 main Diffie–Hellman handshake protocol (left) and key schedule (right). Values L_i and C_i indicate bitstring labels, resp. constant values, (distinct per i). Boxes Ext and Exp denote HKDF extraction resp. expansion, dashed inputs to Exp indicating context information (see protocol figure for detailed computations).

and DH values (within CH‖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). 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 HKDF.Expand with handshake context up to the ServerFinished message. The master secret in turn is derived through (context-less) Expand and Extract from the handshake secret HS.

Handling the TLS 1.3 key schedule. What crucially differentiates the TLS 1.3 handshake from the basic SIGMA-I design is the way keys are derived. While SIGMA-I derives its 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 HS and MS secrets are derived *without* further context purely from the shared DH secret DHE $= g^{xy}$. Only when deriving the specific-purpose secrets—handshake traffic keys, MAC keys, and the session key ATS—are the nonces and DH shares add as session-identifying context. To complicate matters even further, this context is hashed and the final session key ATS depends on more messages than just the session-identifying ones. Recall that the original techniques by Cohn-Gordon et al. [19] heavily relies on (exactly) the session identifiers being input together with DH secrets to a random oracle when programming the latter, impeding a more direct application like for SIGMA-I. In their concurrent work, Diemert and Jager [21] 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. As we will show next, we overcome the technical obstacle of this linking while directly modeling HKDF.Extract and HKDF.Expand as individual random oracles, carefully orchestrating the programming of intermediate secrets and session keys and connecting them through constant-time look-ups. This leads to a slightly less excessive use of the random oracle technique and allows us to stay much closer to the structure of TLS 1.3's key schedule.

6 Tighter Security Proof for the TLS 1.3 Handshake

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

Theorem 5. *Let $\mathcal{A}$ be a key exchange security adversary against the TLS 1.3 handshake protocol as specified in Figure 3 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*

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

Here, $nl = 256$ is the nonce length in TLS 1.3, kl is the output length of $\mathsf{RO}_2 =$ HKDF.Expand*, $\mathbb{G}$ is the used Diffie–Hellman group of prime order p, and $q_{\mathrm{S}} \cdot q_{\mathrm{RO}} \leq 2^{kl-3}$.*[5]

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

Proof idea. Let us first outline the core and novel technical steps, before we give some more proof details below; for space reasons we defer the full proof to the full version [20]. We note that as all keys in the SIGMA exchange are derived from the master key mk, which is itself derived from the shared Diffie–Hellman secret, all intermediate keys in TLS 1.3 are derived from the handshake secret HS, which is derived directly from the shared Diffie–Hellman secret DHE. Embedding a DH challenge into all sessions robs the reduction of the ability to compute HS; as in the SIGMA proof, we will need to use the strong DH oracle to detect and program queries that would output an inconsistent value of HS. Since HS is derived without context, a naive method would have to check every input to $\mathsf{HKDF.Extract}$ against the DH shares received by each session, which would however result in a non-tight, quadratic runtime loss.

We instead leverage that the handshake secret HS is an internal value, not exposed by any oracle. The adversary hence cannot detect an inconsistent HS value until it makes the entire chain of queries leading to one of the keys $\mathrm{tk}_{\mathrm{shs}}$, $\mathrm{tk}_{\mathrm{chs}}$, fk_C, fk_S, or ATS used in SEND, REVSESSIONKEY, and TEST responses. Our reduction prudently sets up a separate bidirectional lookup table for each "link" in that chain. The adversary can make the RO queries in the chain in any order; we need only program the last one for consistency, at which time we have seen the session's DH secret, nonces, and group elements as query inputs. Linking the output of one key-derivation step to the input of the next this way, the reduction can identify the relevant sessions using only constant time and linear space. Together with a careful argument that the attacker is unlikely to guess an intermediate chain value, this allows us to treat $\mathsf{HKDF.Extract}$ and $\mathsf{HKDF.Expand}$ as two individual random oracles. Thereby, we stay close to how HKDF is used in TLS 1.3 and obtain two compact strong-DH bounds.

Now we give a more precise view of the structure of our proof, with a particular focus on nonstandard techniques and the critical random oracle programming in the reduction step to the strong Diffie–Hellman problem, handling the complexity of TLS 1.3's key schedule.

Proof. We develop the bound via a series of code-based game hops.

Game 0. The first game G_0 is the key exchange security game (cf. Figure 1) for the TLS 1.3 handshake protocol (Figure 3). So, $\Pr[\mathrm{G}_0 \Rightarrow 1] = \Pr[\mathrm{G}^{\mathsf{KE\text{-}SEC}}_{\mathsf{TLS},\mathcal{A}} \Rightarrow 1]$.

Games 1–4. Over the next four games we ensure the uniqueness of each session's protocol transcript by aborting if an honest session chooses a nonce and DH share that have already been sent or received by another honest session, or if a collision occurs in the hash function H. We limit the probability of nonce and DH share collisions using a union bound, and give a simple reduction $\mathcal{B}_1$ to the collision resistance of the hash function H. We also lazily sample the random oracles RO_1 and RO_2 using internal tables H_1 and H_2. Excluding collisions, we obtain the bound $\Pr[\mathrm{G}_0 \Rightarrow 1] - \Pr[\mathrm{G}_4 \Rightarrow 1] \leq \frac{3q_S^2}{2^{nl+1} \cdot p} + \mathsf{Adv}^{\mathsf{CR}}_{\mathsf{H}}(t_1)$.

Games 5–6. Following the technique of [19], we let initiator sessions in category (A) copy session, MAC, and traffic encryption keys from their partners via

a table indexed by session IDs. In TLS 1.3, there are two encryption keys $\mathrm{tk}_{\mathrm{shs}}$ and $\mathrm{tk}_{\mathrm{chs}}$, and two MAC keys fk_S and fk_C to copy. One significant difference from both [19] and our SIGMA-I proof is that the session key ATS now depends on the messages $\mathtt{SCRT}$, $\mathtt{SCV}$, and $\mathtt{SF}$. We have not yet ensured that partnered sessions agree on these values. Therefore honest initiators will only copy ATS from their partners if they received the exact same $(\mathtt{SCRT}, \mathtt{SCV}, \mathtt{SF})$ sent by their partner, which they check via an internal look-up table. Otherwise, ATS is still computed as in previous games. Since keys are only copied when partners agree on all of the information entering the key derivation function, this change is unobservable to $\mathcal{A}$, hence $\Pr[\mathrm{G}_6 \Rightarrow 1] = \Pr[\mathrm{G}_4 \Rightarrow 1]$.

Games 7–8. These two games contain both the most critical step and the one that diverges the most from the SIGMA-I proof. We let all category (A) sessions that are not already copying their keys pick the handshake traffic keys SHTS and CHTS, and the session key ATS uniformly at random, checking for consistency with the random oracle RO_2 and retroactively programming it when necessary. (Category (A) initiator sessions who do not copy ATS due to tampering sample only ATS.) Then, we eliminate the consistency check and let these sessions' handshake traffic keys and session key be uniformly random and inconsistent with the adversary's queries to RO_2. We argue that the adversary can only detect this inconsistency if it queries RO_2 on the correct input to derive one of SHTS, CHTS, or ATS for a category (A) session, an event we refer to as event F.

We give a reduction $\mathcal{B}_2$ to the strong DH assumption in group $\mathbb{G}$ which wins with high probability if event F occurs. Given a challenge C, D, algorithm $\mathcal{B}_2$ simulates Game 7. It embeds C in the DH shares of all initiators and D in the DH shares of all category (A) responders. Because $\mathcal{B}_2$ cannot compute the DH secret for embedded sessions, it uses its stDH oracle to catch and program all queries to RO_2 which are dependent on this secret. When event F occurs, $\mathcal{B}_2$ uses its own randomness to extract the challenge DH secret from the DH secret contained in the query that triggered event F. In addition to the details covered in Section 6, the reduction has a few nuances:

1. If for some category (A) session, $\mathcal{A}$ can guess without making the corresponding query any of the intermediate values $\mathrm{HS} = \mathsf{RO}_1(\mathtt{C}_1, \mathrm{DHE})$, $\mathrm{dHS} = \mathsf{RO}_2(\mathrm{HS}, \mathtt{L}_3, \mathsf{H}(\text{“”}))$, or $\mathrm{MS} = \mathsf{RO}_1(\mathrm{dHS}, 0)$, 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. Without knowing DHE, $\mathcal{B}_2$ cannot detect this query, so it does not program RO_2 appropriately and the simulation fails. $\mathcal{B}_2$ does not itself compute HS, dHS, or MS for category (A) sessions, so if $\mathcal{A}$ does not make the appropriate queries than all three values are uniformly random and each can be guessed with probability at most $\frac{q_{\mathrm{RO}} \cdot q_{\mathrm{S}}}{2^{kl}}$.
2. In TLS 1.3, the context string including the Diffie–Hellman shares is hashed with H before it enters the key derivation, so $\mathcal{B}_2$ cannot directly associate an RO_2 query with an honest *sid*. We address this by logging hash computations of honest sessions in a reverse look-up table R. Then in the RO_2 oracle, $\mathcal{B}_2$ can use R to efficiently find the context associated with a particular query.

When $q_{\mathrm{RO}} \cdot q_{\mathrm{S}} \leq 2^{kl-3}$, we obtain the bound $\Pr[\mathrm{G}_6 \Rightarrow 1] - \Pr[\mathrm{G}_8 \Rightarrow 1] \leq 2 \cdot \mathsf{Adv}^{\mathsf{stDH}}_{\mathbb{G}}(t_{\mathcal{B}_2}, q_{\mathrm{RO}}) + \frac{q_{\mathrm{RO}} \cdot q_{\mathrm{S}}}{2^{kl}}$.

The reduction $\mathcal{B}_2$ queries the stDH oracle at most once for each query to RO_2 query and once more when event F occurs. Computing the input to each stDH query requires 1 multiplication and one exponentiation in the base group, which can be done using $1 + 2\log_2 p$ total group operations. In our runtime analysis, we count each group operation as 1 step, so $t_{\mathcal{B}_2} \approx t + 2q_{\mathrm{RO}} \log_2 p$.

Game 9. In game G_9, category (A) sessions sample all encryption and MAC keys uniformly at random. This is distinguishable only if the adversary can query RO on a string containing one of the random values SHTS or CHTS, so by the birthday bound $\Pr[\mathrm{G}_8 \Rightarrow 1] - \Pr[\mathrm{G}_9 \Rightarrow 1] \leq \frac{q_{\mathrm{RO}} \cdot q_{\mathrm{S}}}{2^{kl}}$.

Games 10–13. In the remaining games, we eliminate signature and MAC forgeries via straightforward reductions $\mathcal{B}_3$ and $\mathcal{B}_4$ to the multi-user EUF-CMA security of S and M. This gives the bound $\Pr[\mathrm{G}_9 \Rightarrow 1] - \Pr[\mathrm{G}_{13} \Rightarrow 1] \leq \mathsf{Adv}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{S}}(t_{\mathcal{B}_3, q_{\mathrm{Nw}}, q_{\mathrm{S}}, q_{\mathrm{S}}, q_{\mathrm{RL}}}) + \mathsf{Adv}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{M}}(t_{\mathcal{B}_4}, q_{\mathrm{S}}, q_{\mathrm{S}}, 1, q_{\mathrm{S}}, 1, 0)$.

Finally, we argue that $\mathcal{A}$ has advantage 0 in game G_{13}, using logic similar to that in our SIGMA-I proof, with two slight differences: 1. Partnered sessions no longer use labels to distinguish their MAC tags; instead we note that messages tagged by initiator sessions are strictly longer than messages tagged by responder sessions. 2. We cannot immediately conclude that partnered sessions agree on the same session key because the session key ATS relies on values that are not contained in the session identifier. However, since we have excluded MAC forgeries, all the information entering the derivation of ATS is authenticated by the responder session's MAC tag. □

7 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[6] ($\approx 2^{27}$).

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

Number of sessions $\#S \approx q_{\mathrm{S}} \in \{2^{35}, 2^{45}, 2^{55}\}$**.** Chrome and Firefox 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.[7] 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 TLS 1.3 (bit-security b, 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 groups, 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 DDH 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 logarithm.

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.

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 [17] and for TLS 1.3 by Dowling et al. [24]. We report them in Appendix A 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. We report the numerical advantage bounds for SIGMA and TLS 1.3 based on prior (CK [17], DFGS [24]) 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.

[7] `https://transparencyreport.google.com/`, `https://telemetry.mozilla.org/`

Adv. resources						SIGMA		TLS 1.3	
t	$\#U$	$\#S$	$\#RO$	Curve (bit sec. b, order p)	Target $t/2^b$	CK [17]	Us (Thm. 4)	DFGS [24]	Us (Thm. 5)
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. Advantages of a key exchange adversary with given resources in breaking the security of the SIGMA and TLS 1.3 protocols. See Section 7 for further details.

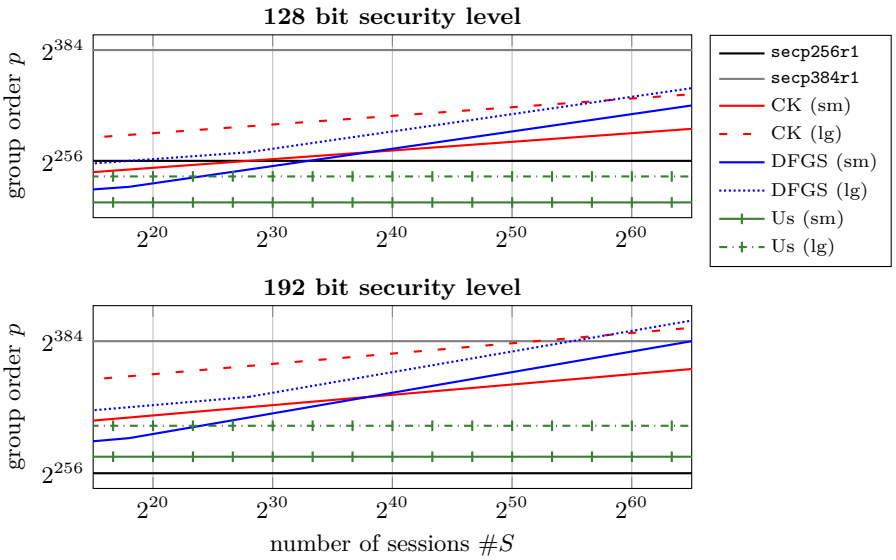

Fig. 4. Elliptic curve group order (y axis) required to achieve 128-bit (top) and 192-bit (bottom) AKE security for SIGMA and TLS 1.3 based on the CK [17] SIGMA, DFGS [24] TLS 1.3, and our bounds (ours giving the same result for SIGMA and TLS 1.3), for a varying number of sessions $\#S$ (x axis). Both 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.

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 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 [17] and DFGS [24] 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 [21] 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. Finally, let us take a slightly different perspective on what the prior and our bounds tell us: Figure 7 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.

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).

References

1. Abdalla, M., Bellare, M., Rogaway, P.: The oracle Diffie-Hellman assumptions and an analysis of DHIES. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 143–158. Springer, Heidelberg (Apr 2001)
2. Abdalla, M., Benhamouda, F., MacKenzie, P.: Security of the J-PAKE password-authenticated key exchange protocol. In: 2015 IEEE Symposium on Security and Privacy. pp. 571–587. IEEE Computer Society Press (May 2015)
3. Abdalla, M., Fouque, P.A., Pointcheval, D.: Password-based authenticated key exchange in the three-party setting. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 65–84. Springer, Heidelberg (Jan 2005)
4. Bader, C., Hofheinz, D., Jager, T., Kiltz, E., Li, Y.: Tightly-secure authenticated key exchange. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015, Part I. LNCS, vol. 9014, pp. 629–658. Springer, Heidelberg (Mar 2015)
5. Bader, C., Jager, T., Li, Y., Schäge, S.: On the impossibility of tight cryptographic reductions. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 273–304. Springer, Heidelberg (May 2016)

6. Bellare, M., Bernstein, D.J., Tessaro, S.: Hash-function based PRFs: AMAC and its multi-user security. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part I. LNCS, vol. 9665, pp. 566–595. Springer, Heidelberg (May 2016)
7. Bellare, M., Canetti, R., Krawczyk, H.: Pseudorandom functions revisited: The cascade construction and its concrete security. In: 37th FOCS. pp. 514–523. IEEE Computer Society Press (Oct 1996)
8. Bellare, M., Dai, W.: The multi-base discrete logarithm problem: Non-rewinding proofs and improved reduction tightness for identification and signatures. In: INDOCRYPT 2020 (2020), https://eprint.iacr.org/2020/416
9. Bellare, M., Goldreich, O., Mityagin, A.: The power of verification queries in message authentication and authenticated encryption. Cryptology ePrint Archive, Report 2004/309 (2004), http://eprint.iacr.org/2004/309
10. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (May 2000)
11. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) ACM CCS 93. pp. 62–73. ACM Press (Nov 1993)
12. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO'93. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (Aug 1994)
13. Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 409–426. Springer, Heidelberg (May / Jun 2006)
14. Boneh, D.: The decision Diffie-Hellman problem. In: Third Algorithmic Number Theory Symposium (ANTS). LNCS, vol. 1423. Springer, Heidelberg (1998), invited paper
15. Brendel, J., Fischlin, M., Günther, F., Janson, C.: PRF-ODH: Relations, instantiations, and impossibility results. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part III. LNCS, vol. 10403, pp. 651–681. Springer, Heidelberg (Aug 2017)
16. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 453–474. Springer, Heidelberg (May 2001)
17. Canetti, R., Krawczyk, H.: Security analysis of IKE's signature-based key-exchange protocol. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 143–161. Springer, Heidelberg (Aug 2002), http://eprint.iacr.org/2002/120/
18. Cohn-Gordon, K., Cremers, C., Garratt, L.: On post-compromise security. In: 2016 Computer Security Foundations Symposium. pp. 164–178. IEEE (2016)
19. Cohn-Gordon, K., Cremers, C., Gjøsteen, K., Jacobsen, H., Jager, T.: Highly efficient key exchange protocols with optimal tightness. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 767–797. Springer, Heidelberg (Aug 2019)
20. Davis, H., Günther, F.: Tighter proofs for the SIGMA and TLS 1.3 key exchange protocols. Cryptology ePrint Archive, Report 2020/1029 (2020), https://eprint.iacr.org/2020/1029
21. Diemert, D., Jager, T.: On the tight security of TLS 1.3: Theoretically-sound cryptographic parameters for real-world deployments. Journal of Cryptology (2020), to appear. Available as Cryptology ePrint Archive, Report 2020/726. https://eprint.iacr.org/2020/726

22. Dowling, B., Fischlin, M., Günther, F., Stebila, D.: A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015. pp. 1197–1210. ACM Press (Oct 2015)
23. Dowling, B., Fischlin, M., Günther, F., Stebila, D.: A cryptographic analysis of the TLS 1.3 draft-10 full and pre-shared key handshake protocol. Cryptology ePrint Archive, Report 2016/081 (2016), http://eprint.iacr.org/2016/081
24. Dowling, B., Fischlin, M., Günther, F., Stebila, D.: A cryptographic analysis of the TLS 1.3 handshake protocol. Journal of Cryptology (2021), to appear. Available as Cryptology ePrint Archive, Report 2020/1044. https://eprint.iacr.org/2020/1044
25. Fischlin, M., Günther, F.: Multi-stage key exchange and the case of Google's QUIC protocol. In: Ahn, G.J., Yung, M., Li, N. (eds.) ACM CCS 2014. pp. 1193–1204. ACM Press (Nov 2014)
26. Fischlin, M., Günther, F.: Replay attacks on zero round-trip time: The case of the TLS 1.3 handshake candidates. In: 2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017. pp. 60–75. IEEE (Apr 2017)
27. Gjøsteen, K., Jager, T.: Practical and tightly-secure digital signatures and authenticated key exchange. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 95–125. Springer, Heidelberg (Aug 2018)
28. Harkins, D., Carrel, D.: The Internet Key Exchange (IKE). IETF RFC 2409 (Proposed Standard) (1998)
29. Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: On the security of TLS-DHE in the standard model. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 273–293. Springer, Heidelberg (Aug 2012)
30. Kaufman (Ed.), C.: Internet Key Exchange (IKEv2) Protocol. RFC 4306 (Proposed Standard) (Dec 2005), https://www.rfc-editor.org/rfc/rfc4306.txt, obsoleted by RFC 5996, updated by RFC 5282
31. Kent, S., Atkinson, R.: Security Architecture for the Internet Protocol. RFC 2401 (Proposed Standard) (Nov 1998), https://www.rfc-editor.org/rfc/rfc2401.txt, obsoleted by RFC 4301, updated by RFC 3168
32. Krawczyk, H.: SIGMA: The "SIGn-and-MAc" approach to authenticated Diffie-Hellman and its use in the IKE protocols. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 400–425. Springer, Heidelberg (Aug 2003)
33. Krawczyk, H.: SIGMA: the 'SIGn-and-MAc' approach to authenticated Diffie-Hellman and its use in the IKE protocols (2003), full version. https://webee.technion.ac.il/~hugo/sigma-pdf.pdf
34. Krawczyk, H.: Cryptographic extraction and key derivation: The HKDF scheme. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 631–648. Springer, Heidelberg (Aug 2010)
35. Krawczyk, H., Wee, H.: The OPTLS protocol and TLS 1.3. In: 2016 IEEE European Symposium on Security and Privacy. pp. 81–96. IEEE (Mar 2016)
36. LaMacchia, B.A., Lauter, K., Mityagin, A.: Stronger security of authenticated key exchange. In: Susilo, W., Liu, J.K., Mu, Y. (eds.) ProvSec 2007. LNCS, vol. 4784, pp. 1–16. Springer, Heidelberg (Nov 2007)
37. Langley, A., Hamburg, M., Turner, S.: Elliptic Curves for Security. RFC 7748 (Informational) (Jan 2016), https://www.rfc-editor.org/rfc/rfc7748.txt
38. Maurer, U.M.: Abstract models of computation in cryptography (invited paper). In: Smart, N.P. (ed.) 10th IMA International Conference on Cryptography and Coding. LNCS, vol. 3796, pp. 1–12. Springer, Heidelberg (Dec 2005)
39. National Institute of Standards and Technology: FIPS PUB 186-4: Digital Signature Standard (DSS) (2013)

40. Okamoto, T., Pointcheval, D.: The gap-problems: A new class of problems for the security of cryptographic schemes. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 104–118. Springer, Heidelberg (Feb 2001)
41. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Proposed Standard) (Aug 2018), `https://www.rfc-editor.org/rfc/rfc8446.txt`
42. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT'97. LNCS, vol. 1233, pp. 256–266. Springer, Heidelberg (May 1997)

A Evaluation Details

Fully-quantitative CK SIGMA Bound. Comparing our SIGMA bound from Theorem 4 to the original security proof by Canetti and Krawczyk [17] (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 [16] 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 [16] in the same style as our AKE model (cf. Definition 1), capturing the same oracles plus an additional state-reveal oracle, with q_{RST} 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 [17, Theorem 6 in the full version], we obtained the following concrete security bound:

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

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 in [17, 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 $q_{\mathrm{NW}} = 1$, which are equivalent to the corresponding single-user advantages (cf. Appendix B).

Fully-quantitative DFGS TLS 1.3 Bound. We compare our security bound for TLS 1.3 from Theorem 5 with the bound of Dowling et al. [24] (DFGS).

Note that this bound is established in a multi-stage key exchange model [25], 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 HKDF.Extract) while other PRF steps remain in the standard model. Our proof instead models both HKDF.Extract and HKDF.Expand as random oracles. Translating the bound from [24, Theorems 5.1, 5.2] yields:

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

Let us further unpack the PRF-ODH term. Following Brendel et al. [15], it can be reduced to the strong Diffie–Hellman assumption modeling HKDF.Extract as a random oracle.[8] 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: $\mathsf{Adv}^{\mathsf{dual\text{-}snPRF\text{-}ODH}}_{\mathsf{RO},\mathbb{G}}(t_{\mathcal{B}_3}, q_{\mathrm{RO}}) \leq \mathsf{Adv}^{\mathsf{stDH}}_{\mathbb{G}}(t_{\mathcal{B}_3}, q_{\mathrm{RO}})$.

B Assumptions, Building Blocks, Multi-User Security

Definition 6 (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* $\mathrm{G}^{\mathsf{mu\text{-}PRF}}_{\mathsf{PRF},\mathcal{A}}$ *be the multi-user PRF security game defined as in Figure 5. We define* $\mathsf{Adv}^{\mathsf{mu\text{-}PRF}}_{\mathsf{PRF}}(t, q_{\mathrm{NW}}, q_{\mathrm{FN}}, q_{\mathrm{FN/U}}) := 2 \cdot \max_{\mathcal{A}} \Pr\left[\mathrm{G}^{\mathsf{mu\text{-}PRF}}_{\mathsf{PRF},\mathcal{A}} \Rightarrow 1\right] - 1$*, where the maximum is taken over all adversaries, denoted* $(t, q_{\mathrm{NW}}, q_{\mathrm{FN}}, q_{\mathrm{FN/U}})$-mu-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* $\mathrm{Fn}(i, \cdot)$ *for any user* i*.*

Generically, the multi-user security of PRFs reduces to single-user security (formally, $\mathrm{G}^{\mathsf{mu\text{-}PRF}}_{\mathsf{PRF},\mathcal{A}}$ with $\mathcal{A}$ restricted to $q_{\mathrm{NW}} = 1$ queries to New) with a factor in the number of users via a hybrid argument [7], i.e., $\mathsf{Adv}^{\mathsf{mu\text{-}PRF}}_{\mathsf{PRF}}(t, q_{\mathrm{NW}}, q_{\mathrm{FN}}, q_{\mathrm{FN/U}}) \leq q_{\mathrm{NW}} \cdot \mathsf{Adv}^{\mathsf{mu\text{-}PRF}}_{\mathsf{PRF}}(t', 1, q_{\mathrm{FN/U}}, q_{\mathrm{FN/U}})$, 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 [6], that however achieve multi-user security tightly. If we use a random oracle RO as a PRF with key length kl, then $\mathsf{Adv}^{\mathsf{mu\text{-}PRF}}_{\mathsf{RO}}(t, q_{\mathrm{NW}}, q_{\mathrm{FN}}, q_{\mathrm{FN/U}}, q_{\mathrm{RO}}) \leq \frac{q_{\mathrm{NW}} \cdot q_{\mathrm{RO}}}{2^{kl}}$.

[8] 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.

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

$\underline{\textsc{Initialize}:}$
1 $b \xleftarrow{\$} \{0,1\}$
2 $u \leftarrow 0$

$\underline{\textsc{New}:}$
3 $u \leftarrow u + 1$
4 if $b = 1$ then
5 $\quad K_u \xleftarrow{\$} \{0,1\}^k$; $f_u := \mathsf{PRF}(K_u, \cdot)$
6 else $f_u \xleftarrow{\$} \mathsf{FUNC}$

$\underline{\textsc{Fn}(i,x)}$:
7 return $f_i(x)$

$\underline{\textsc{Finalize}(b^*)}$:
8 return $[[b = b^*]]$

Fig. 5. 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 7 (Signature mu-EUF-CMA security [4]). *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 (see the full version [20] 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* $\textsc{Sign}(i, \cdot)$ *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 [5].

Definition 8 (MAC mu-EUF-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 (see the full version [20] 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* $(t, q_{\mathrm{NW}}, q_{\mathrm{TG}}, q_{\mathrm{TG/U}}, q_{\mathrm{VF}}, q_{\mathrm{VF/U}}, q_{\mathrm{C}})$-mu-EUF-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* $\textsc{Tag}(i, \cdot)$*, resp.* $q_{\mathrm{VF/U}}$ *queries* $\textsc{Vrfy}(i, \cdot)$ *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: $\mathsf{Adv}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{M}}(t, q_{\mathrm{NW}}, q_{\mathrm{TG}}, q_{\mathrm{TG/U}}, q_{\mathrm{VF}}, q_{\mathrm{VF/U}}, q_{\mathrm{C}}) \leq q_{\mathrm{NW}} \cdot \mathsf{Adv}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{M}}(t, 1, q_{\mathrm{TG/U}}, q_{\mathrm{TG/U}}, q_{\mathrm{VF/U}}, q_{\mathrm{VF/U}}, 0)$, 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 [9] 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 [9].

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 [6] 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 $\mathsf{Adv}^{\mathsf{mu\text{-}EUF\text{-}CMA}}_{\mathsf{RO}}(t, q_{\mathrm{NW}}, q_{\mathrm{TG}}, q_{\mathrm{TG/U}}, q_{\mathrm{VF}}, q_{\mathrm{VF/U}}, q_{\mathrm{C}}, q_{\mathrm{RO}}) \leq \frac{q_{\mathrm{VF}}}{2^{ol}} + \frac{(q_{\mathrm{NW}} - q_{\mathrm{C}}) \cdot q_{\mathrm{RO}}}{2^{kl}}$.

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

If we use a random oracle RO as hash function, then $\mathsf{Adv}^{\mathsf{CR}}_{\mathsf{RO}}(t, q_{\mathrm{RO}}) \leq \frac{q_{\mathrm{RO}}^2}{2^{ol+1}} + \frac{1}{2^{ol}}$.

C Proof of the Strong Diffie–Hellman GGM Bound

We establish the bound of Theorem 3 through a sequence of incrementally changing code-based games; see the full version [20] for complete details.

Game 0. We formalize the strong Diffie–Hellman problem in the GGM using the setting and notation of Bellare and Dai [8]. Briefly, we represent a group a group of prime order p using an arbitrary set $\mathbb{G}$ of label strings and a randomly chosen bijection $E : \mathbb{Z}_p \to \mathbb{G}$, called the encoding function. For any two strings $A, B \in \mathbb{G}$, we define the operation $A \operatorname{OP}_E B = E(E^{-1}(A) + E^{-1}(B) \mod p)$. The adversary is given the identity element $\mathbb{1} = E(0)$, a generator $g = E(1)$, challenges X and Y, and oracle access to OP_E through an oracle OP. Note that for any integer $a \in \mathbb{Z}_p$, we can compute $g^a = E(a)$. On an input A, B, the stDH oracle uses this property to find the discrete logarithm a of A in order to check whether $E(xa) = X^a = B$. Throughout, we track the set GL of group element labels the adversary has seen, and return $\perp$ in response to all oracle queries containing other labels. By definition, $\mathsf{Adv}^{\mathsf{stDH}}_{\mathbb{G}}(t, q_{\mathrm{sDH}}) = \Pr[\mathrm{G}_0 \Rightarrow 1]$.

Game 1. Although the notation of G_0 is simpler and more intuitive, it is more useful for the proof game to internally represent elements of $\mathbb{G}$ with vectors over $\mathbb{Z}_p^3$ instead of integers in $\mathbb{Z}_p$, as we do in Figure 6. We map elements $\vec{t} \in \mathbb{Z}_p^3$ back to $\mathbb{Z}_p$ by taking the inner product of $\vec{t}$ with the vector $(1, x, y)$. (Effectively, we take $\vec{t}$ to be the coefficients of a linear combination of 1, x, and y, which are represented respectively by the basis vectors $\vec{e_1}$, $\vec{e_2}$, and $\vec{e_3}$.)

Composing this map with the encoding function E induces a transformation from $\mathbb{Z}_p^3$ to $\mathbb{G}$, which we implement via an internal oracle VE. We cache the transformation in table TV and its inverse in table TI. Each element of $\mathbb{G}$ now has multiple vector representations, but the bilinearity of the inner product ensures that the view of the adversary is not changed, and $\Pr[\mathrm{G}_1] = \Pr[\mathrm{G}_0]$.

Games 2–3. In Game G_3, we make two undetectable changes: we lazily sample the bijection E, and in the stDH oracle, we replace the condition $\mathrm{VE}(x\vec{a}) = B = \mathrm{VE}(\vec{b})$ with the equivalent condition $\langle x\vec{a} - \vec{b}, \vec{x} \rangle = 0$.

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

$\underline{\textsc{Initialize}():}$

1 $p \leftarrow |\mathbb{G}|$; $E \xleftarrow{\$} \text{Bijections}(\mathbb{Z}_p, \mathbb{G})$
2 $\mathbb{1} \leftarrow \mathrm{VE}(\vec{0})$; $g \leftarrow \mathrm{VE}(\vec{e_1})$
3 $x, y \xleftarrow{\$} \mathbb{Z}_p^*$; $\vec{x} \leftarrow 1, x, y$
4 $X \leftarrow \mathrm{VE}(\vec{e_2})$; $Y \leftarrow \mathrm{VE}(\vec{e_3})$
5 return $(\mathbb{1}, g, x, y)$

$\underline{\mathrm{OP}(A, B, \text{ sgn }):}$

6 $\vec{c} \leftarrow TI(A) \text{ sgn } TI(B) \mod p$
7 $C \leftarrow \mathrm{VE}(\vec{c})$; return C

$\underline{\mathrm{VE}(\vec{t}):}$

1 if $TV[\vec{t}] \neq \perp$ then return $TV[\vec{t}]$
2 $v \leftarrow \langle \vec{t}, \vec{x} \rangle$; $TV[\vec{t}] \leftarrow E(v)$;
3 $GL \leftarrow GL \cup \{TV[\vec{t}]\}$; $TI[TV[\vec{t}]] \leftarrow \vec{t}$
4 return $TV[\vec{t}]$

$\underline{\mathsf{stDH}(A, B):}$

8 $\vec{a} \leftarrow TI(A)$; $\vec{b} \leftarrow TI(B)$
9 return $[[\mathrm{VE}(x\vec{a}) = B]]$

$\underline{\textsc{Finalize}(Z):}$

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

Fig. 6. Game G_1 of the stDH proof.

We continue in the next game by sampling the entries of TV directly instead of through calls to E. Distinct vectors $\vec{t}$ and $\vec{t'}$ no longer map to the same group element when $\langle \vec{t}, \vec{x} \rangle = \langle \vec{t'}, \vec{x} \rangle$. The adversary cannot notice this change unless two such t, t' are queried to VE; we call this event F_1 and let Finalize return true when it occurs. This only increases the success probability of the adversary, so $\Pr[\mathrm{G}_1] \leq \Pr[\mathrm{G}_3]$. At this point, function E is unused and becomes redundant.

Game 4. The adversary can trivially get a true response from the stDH oracle by computing $A = g^a$ for any integer a and $B = X^a$. We now return false in all other cases. Let F_2 be the event where the adversary makes a nontrivial query (A, B) to stDH that should return true, i.e., one where $\langle xTI(A) - TI(B), \vec{x} \rangle = 0$. Unless F_2 occurs, the output of stDH does not change, so $\Pr[\mathrm{G}_3] \leq \Pr[\mathrm{G}_4 \text{ and } \overline{F_2}] + \Pr[F_2]$.

Game 5. This game is identical to G_4, except Finalize returns true whenever event F_2 could have occurred. It follows that $\Pr[\mathrm{G}_3] \leq \Pr[\mathrm{G}_5]$. At this point, variables x, y, and $\vec{x}$ are not used by any oracle except Finalize, so we delay their initialization until the end of the game without detection by the adversary.

Collecting bounds reveals that $\mathsf{Adv}^{\mathsf{stDH}}_{\mathbb{G}}(t, q_{\mathrm{sDH}}) \leq \Pr[\mathrm{G}_5]$. A t-query adversary playing G_5 wins only if events F_1 or F_2 occur, or if $[\mathrm{VE}(x\vec{e_3}) = Z]$. Event F_1 occurs when table TI contains distinct $\vec{t_i}, \vec{t_j}$ such that $\langle \vec{t_i} - \vec{t_j}, (1, x, y) \rangle$. This means (x, y) is a root of the bivariate linear polynomial $(\vec{t_i} - \vec{t_j})[0] + (\vec{t_i} - \vec{t_j})[1] \cdot x + (\vec{t_i} - \vec{t_j})[2] \cdot y$. Since x and y are sampled independently by the Finalize oracle, this occurs with probability at most $1/p$ for each polynomial by Lemma 1 of [42]. Event F_2 occurs when $\langle x\vec{t_i} - \vec{t_j}, \vec{x} \rangle = 0$ for some t_i, t_j in TI. Similarly, this means that (x, y) must be a root of the quadratic $(x\vec{t_i} - \vec{t_j})[0] + (x\vec{t_i} - \vec{t_j})[1] \cdot x + (x\vec{t_i} - \vec{t_j})[2] \cdot y$. By Lemma 1, this occurs with probability at most $2/p$ for each $(\vec{t_i}, \vec{t_j})$ pair. Finally, $[\mathrm{VE}(x\vec{e_3}) = Z]$ holds with probability at most $1/p$ because $\mathrm{VE}(x\vec{e_3})$ is uniformly random.

Taking a union bound over the $(t + 4)^2$ possible pairs $(\vec{t_i}, \vec{t_j})$, we obtain $\Pr[\mathrm{G}_5] \leq (3(t + 4)^2 + 1)/p$. The theorem statement follows for all $t > 25$. □