An abridged version of this paper was published at IEEE Symposium on Security and Privacy 2016.

Downgrade Resilience in Key-Exchange Protocols

Karthikeyan Bhargavan[*] Christina Brzuska[†]
Cédric Fournet, Markulf Kohlweiss, Santiago Zanella-Béguelin[‡]
Matthew Green[§]

January 2016

Abstract

Key-exchange protocols such as TLS, SSH, IPsec, and ZRTP are highly configurable, with typical deployments supporting multiple protocol versions, cryptographic algorithms and parameters. In the first messages of the protocol, the peers negotiate one specific combination: the *protocol mode*, based on their local configurations. With few notable exceptions, most cryptographic analyses of configurable protocols consider a single mode at a time. In contrast, downgrade attacks, where a network adversary forces peers to use a mode weaker than the one they would normally negotiate, are a recurrent problem in practice.

How to support configurability while at the same time guaranteeing the preferred mode is negotiated? We set to answer this question by designing a formal framework to study downgrade resilience and its relation to other security properties of key-exchange protocols. First, we study the causes of downgrade attacks by dissecting and classifying known and novel attacks against widely used protocols. Second, we survey what is known about the downgrade resilience of existing standards. Third, we combine these findings to define downgrade security, and analyze the conditions under which several protocols achieve it. Finally, we discuss patterns that guarantee downgrade security by design, and explain how to use them to strengthen the security of existing protocols, including a newly proposed draft of TLS 1.3.

[*]Inria Paris-Rocquencourt. E-mail: `karthikeyan.bhargavan@inria.fr`

[†]Hamburg University of Technology. E-mail: `brzuska@tuhh.de`. Part of this work was done at Microsoft Research.

[‡]Microsoft Research. E-mail: `{fournet,markulf,santiago}@microsoft.com`

[§]Johns Hopkins Univeristy. E-mail: `mgreen@cs.jhu.de`

Contents

1 Introduction

Popular protocols such as TLS, SSH and IPSec as used in practice do not fit a simple textbook definition of a key-exchange protocol, where the state machine, cryptographic algorithms, parameters and message formats are all fixed in advance. Rather, these modern protocols feature cryptographic agility, which provides for configurable selection of multiple protocol and cipher modes, so that the key exchange actually executed between two peers depends on a negotiation phase embedded in the exchange.

Agility has proven important in securing real-world protocol implementations. For example, in the wake of recent vulnerability disclosures in TLS [26, 2–4, 48, 27, 12], network operators reacted by updating client and server configurations to disable weak algorithms and protocol versions. Moreover, experience shows that when sufficient agility is not present within a single protocol, application developers construct their own *ad hoc* negotiation mechanisms, for example, by sequentially attempting connections with different versions of a protocol and "falling back" to the best one supported [46].

Unfortunately, support for algorithm agility opens up opportunities for *downgrade attacks*, where an active network adversary interferes with the negotiation, causing honest peers to complete a key exchange, albeit using a mode that is weaker than the one they would have used on their own. Such attacks have been identified in a number of protocols, most famously in the early versions of the SSL protocol [52] and even in recent versions of TLS [2, 46].

Surprisingly, there has been relatively little formal work around the security of negotiation in modern cryptographic protocols. Several recent works formally prove the security of different aspects of TLS and SSH. Some [29, 37] only model a single mode at a time. Some [14, 15] do model negotiation of *weak* algorithms, but do not guarantee negotiation of the preferred mode. Some others [11, 24] consider only interactions where both parties have secure configurations. For this reasons, all of these works overlook certain downgrade attacks that occur when one party supports an insecure mode.

This is concerning because negotiation has proven to be fertile ground for attacks, e.g. [12, 52, 2], and because recent Internet-wide scans have revealed the prevalence of hosts supporting insecure protocol modes [51, 2]. In this setting, it is insufficient to restrict our attention to situations where both parties support secure configurations.

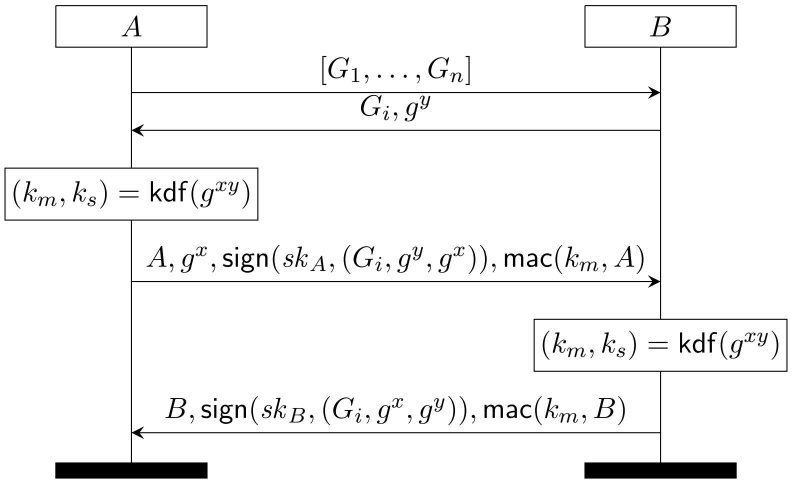
In this work we aim to address this situation by systematically investigating the problem of *downgrade resilience* in cryptographic protocols.

1.1 Motivating example

We begin with a simple motivating example: we adapt the SIGMA protocol of Krawczyk [36] by adding a naïve extension intended to negotiate Diffie-Hellman groups: In the first message, A proposes a list of groups it supports; in the second message, B indicates which of these groups should be used in the exchange. The modified protocol appears in Fig. 1. The goal of the protocol is to compute session keys (k_m, k_s).

Under normal circumstances, the protocol succeeds in correctly selecting a group. However, consider a scenario where both participants support both strong and weak groups. B's signature authenticates the chosen group, but not A's proposal. This leads to a downgrade attack (see Fig. 2) similar to the Logjam attack on TLS [2], where an attacker can break the session keys at leisure and compromise the connection.

Protocol designers have adopted a number of techniques to prevent such downgrade attacks. Based on a review of deployed protocols, we identify three common patterns. In the first, exemplified by SSH, protocol designers assume the existence of strong signing keys shared between the two parties, and use these keys to authenticate all negotiation messages, either at the time they are transmitted, or after the fact.



A
B
$[G_1, \dots, G_n]$
G_i, g^y
$(k_m, k_s) = \mathsf{kdf}(g^{xy})$
$A, g^x, \mathsf{sign}(sk_A, (G_i, g^y, g^x)), \mathsf{mac}(k_m, A)$
$(k_m, k_s) = \mathsf{kdf}(g^{xy})$
$B, \mathsf{sign}(sk_B, (G_i, g^x, g^y)), \mathsf{mac}(k_m, B)$

Figure 1: SIGMA-N: Basic SIGMA [36] with group negotiation

In a second pattern, exemplified by TLS, designers transmit unauthenticated protocol messages, perform a key exchange, and then subsequently use the resulting shared secrets to retroactively authenticate the negotiation messages. The final approach relies on *whitelisting* certain modes, and is best exemplified by Google's TLS False Start proposal [38], which is being codified as part of TLS 1.3 [47].

Each approach has various advantages and disadvantages. The devil is often in the details: each protocol is sensitive to the precise nature of the implementation, e.g. the inputs of authentication functions, or the specifics of what a *valid* mode is for whitelisting. As a concrete example, modern versions of TLS-DHE fail to sign the identity of the ciphersuite chosen by a server, leading to cross-protocol attacks [41, 2]. Similarly, TLS False Start relies solely on ciphersuite identifier (rather than more detailed information such as key strength) in its selection of which modes to whitelist, which converts the online attack of Adrian et al. [2] into an offline one.

1.2 Overview of our approach

We give a definition and a theorem for downgrade resilience that model the following intuitive and desirable property for deployed key-exchange protocols:

> *To prevent an attack on a particular protocol mode, it is sufficient to deactivate the configurations that lead to its negotiation.*

Our work builds on the definitions of Bhargavan et al. [15], used to model security in miTLS, a reference implementation of the TLS standard. A fundamental difference between these definitions and previous work is that they attempt to model entire deployed protocols. This requires a definition of security cognizant of the fact that some aspects (modes) of the protocol may be insecure. To deal with this, the definitions of [15] incorporate predicates determining modes that are expected to provide security guarantees, e.g., key indistinguishability. This approach allows to define security when secure modes are chosen, yet *tolerates* the existence of insecure modes.

One limitation of these definitions is that they do not take into account *how* modes are chosen. In a protocol secure under the miTLS framework, two parties under adversarial influence may arrive at an

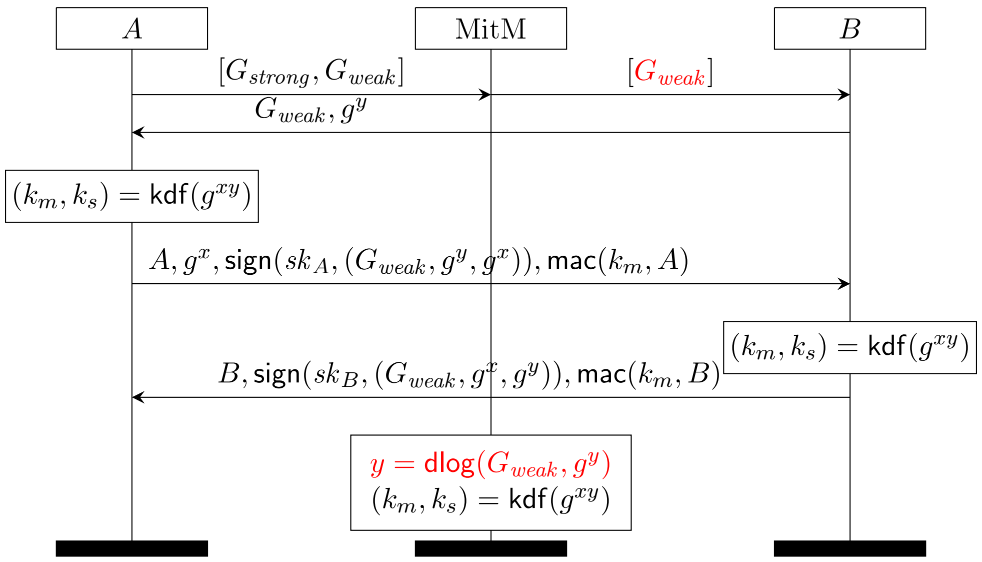

A
MitM
B
$[G_{strong}, G_{weak}]$
$[G_{weak}]$
G_{weak}, g^y
$(k_m, k_s) = \mathsf{kdf}(g^{xy})$
$A, g^x, \mathsf{sign}(sk_A, (G_{weak}, g^y, g^x)), \mathsf{mac}(k_m, A)$
$(k_m, k_s) = \mathsf{kdf}(g^{xy})$
$B, \mathsf{sign}(sk_B, (G_{weak}, g^x, g^y)), \mathsf{mac}(k_m, B)$
$y = \mathsf{dlog}(G_{weak}, g^y)$
$(k_m, k_s) = \mathsf{kdf}(g^{xy})$

Figure 2: Man-in-the-Middle downgrade attack on SIGMA-N

insecure mode even when otherwise they would use a secure mode. In theory each party can detect and react to the negotiation of an insecure mode, e.g., by terminating the protocol execution. Nonetheless, this does not guarantee that the preferred common mode is selected. Our solution is to incorporate *downgrade resilience* in our security definitions, to ensure that an adversary cannot force the selection of another mode than the preferred one.

We consider protocols between an Initiator and a Responder. These two parties each have their own local static *configurations*, expressing their preferences and their intent to negotiate a shared protocol *mode*. To define *downgrade resilience* formally, we introduce a downgrade protection predicate *DP* that operates on pairs of configurations (analogous to miTLS predicates on modes), and that identifies pairs of configurations from which we expect downgrade resilience. We also introduce a function *Nego* that maps two opposite-role configurations to the protocol mode that should be negotiated in the absence of active adversaries. Intuitively, our definition says that a protocol is *downgrade secure* if two peers starting from configurations satisfying *DP* can only negotiate the mode determined by *Nego*, even in the presence of an active adversary.

By way of example, a specific instantiation of *Nego* for the TLS protocol might determine that two TLS peer configurations would normally result in the negotiation of TLS 1.2 in combination with a ciphersuite such as `DHE-RSA-AES256-GCM-SHA384` with a 2048-bit Diffie-Hellman modulus. However, if a server supports an insecure mode, such as a `DHE-EXPORT` ciphersuite, an adversary might force the pair to downgrade to this mode [2]. This shows that without additional countermeasures, TLS 1.2 does not meet our definition. On the other hand, protocols with only one possible mode are obviously secure. The challenge we address in this paper is to consider agile protocols that support multiple modes (e.g., ciphersuites, versions).

To apply our definition to real-world protocols, we adopt the following approach. Rather than analyzing a protocol in its entirety, we first extract a core negotiation sub-protocol, which captures the main downgrade-protection mechanisms of the larger protocol. We next prove that this sub-protocol is *complete* for downgrade security, in the sense that an adversary that succeeds in downgrading the full protocol will also succeed in downgrading the sub-protocol. This separates the analysis into two tasks: one of carefully studying specifications (and possibly in the future implementations) to extract a sub-protocol description and one of analyzing a cryptographic property in a formal model.

We found this technique valuable as the two tasks require quite different skill sets. The former requires an in depth understanding of real-world protocols, the latter a background in provable security. The only other feasible alternative but much more work intensive approach that we are aware of is to analyze a protocol implementation in its entirety as in the miTLS work [15].

Formally, the technique of *lifting* security from the sub-protocol to the main protocol was previously employed by Bergsma et al. [11] to prove multi-ciphersuite security.

In our analysis we restrict ourselves to the manual extraction of sub-protocols that only cover specific families of modes, e.g., signature-based modes or pre-shared key modes, while some of our attacks are cross family attacks. Proving the absence of cross-family attacks requires either to consider more complex sub-protocols that encompass several families, or to study families independently and prove a composition theorem similar to that in Bergsma et al. [11]. Our work is a stepping stone in this direction, and our results are readily applicable in situations where peer configurations are from the same family.

In developing our definitional approach we did not use any automated protocol analysis tools. We could have written machine-checked proofs to show that our sub-protocols correctly abstract full protocols, but doing so requires to write formal descriptions of protocols in excruciating detail, a significant amount of work on its own. Proving properties such as downgrade security is at the limit of what current tools can handle [9]. However costly, we think this is a worthwhile endeavor for standards, and we plan to extend our formalization of TLS in F* to verify our results on downgrade security.

1.3 Summary of our results

Our primary contribution is a novel downgrade security definition for key-exchange protocols. We devise a methodology to analyze the downgrade security of a complex protocol by abstracting away irrelevant details and studying only the core negotiation sub-protocol.

We demonstrate the relevance of our definition and the applicability of our methodology by analyzing the downgrade security of several exemplary real-world protocols, namely TLS, SSH, IPSec and ZRTP. We do so by taking in their standard specifications and extracting appropriate core negotiation sub-protocols. Our analysis identifies known and novel attacks on certain configurations of the protocols as well as sufficient conditions under which these protocols achieve downgrade security. These conditions inform users of these protocols as to how to restrict host configurations to best avoid downgrade attacks.

The following are concrete novel contributions:

- We describe new downgrade vulnerabilities on IKEv2 and ZRTP. These vulnerabilites are present in the protocol standards, but they can be avoided by carefully configured implementations.
- We confirm the conclusion evidenced by recent attacks: TLS versions up to 1.2 are not generally downgrade secure.
- We prove a downgrade security theorem for SSHv2 with `publickey` client authentication that is stronger than previous results. This stems from both peers signing all the messages that determine the protocol mode.
- We show that although TLS 1.3 Draft 10 [47] includes a mandatory server-side message for signing the handshake transcript, this does not prevent downgrades to earlier versions of TLS or non-preferred groups. Informed by this analysis, we define and prove two new downgrade protection mechanisms.

The concrete countermeasures, designed jointly with the core TLS 1.3 working group, have been included in Draft 11.

1.4 Outline of the paper

The remainder of this paper proceeds as follows. In §2 we introduce the terminology used throughout and we provide a primer on security definitions for key exchange protocols. We also introduce downgrade resilience, formally defined in §3. In §4 through §7 we apply these definitions to analyze the security of SSH, IPSec IKE, ZRTP and TLS. We survey related work in §8 and conclude with lessons learned in §9.

2 Modeling Multi-Mode Key-Exchanges

The two main security aspects of popular key-exchange models [15, 11, 25] are entity authentication and key-indistinguishability [10]. Our focus lies on considering multi-mode protocols and incorporating the negotiation of the mode into the security model.

A key exchange protocol is a two-party protocol with an initiator role I and a responder role R (sometimes called client and server). We refer to the protocol, together with an execution model that enables the adversary to interact with multiple sessions of the protocol, as Π. (We will later define security properties as games by combining the execution model with a winning condition for the adversary.) Each session π maintains variables in a local state and makes assignments to them before sending or after receiving a message. We write $\pi.x$ for the value of variable x in session π. We will consider the following variables:

$\pi.cfg$	initial configuration (including the role);
$\pi.uid$	unique identifier of the session;
$\pi.mode$	negotiated *mode* (including long-term identities);
$\pi.key$	session key;
$\pi.complete$	flag set when the session completes successfully.

Variables are initialized to $\perp$ and each session assigns a value to each variable only once, typically in the order given above. The configuration variable $\pi.cfg$ is assigned when a session is created and contains other variables, including one for the session role. We use $\pi.role$ as shorthand for $\pi.cfg.role$ and let $\overline{I} = R$, $\overline{R} = I$.

An adversary interacts with sessions via queries to oracles. A query $\pi \leftarrow \mathsf{Init}(cfg)$ initializes a session. Recall that *cfg* determines *role* and furthermore, in the setting where we have symmetric or public keys, *cfg* will contain handles to those keys. A query $m_{out} \leftarrow \mathsf{Send}(\pi, m_{in})$ sends a message m_{in} to session π, which processes it to update its local state and output an ongoing message m_{out}. A query $k \leftarrow \mathsf{Reveal}(\pi)$ reveals the session key of π, i.e., returns the value of $\pi.key$. There are several variants for handling long-term keys and other authentication mechanisms as well as corruption settings, and each of those requires different variables and oracles. As these settings are mostly standard and orthogonal to our definition, we leave those details deliberately unspecified for now and get back to them in Section 4. Note that our definitions only become complete once we add the specifics of long-term keys or other authentication mechanisms.

2.1 Unique identifiers and partnering

The goal of a key exchange protocol is to *match* two sessions of two different parties so that they compute the same key and agree on the algorithms and authentication setting. We say that two sessions match if and only if they derive the same session key [35]. For defining downgrade security, we rely on the weaker notion of *partnering*, based on unique identifiers—*at most* two sessions may assign the same value to the session variable *uid*.

Definition 1 (Partnering)**.** *Sessions π and π' are partnered if $\pi'.role = \overline{\pi.role}$ (they have opposite roles) and $\pi'.uid = \pi.uid$. A session π is unpartnered when there is no such π'.*

Definition 2 (Uniqueness)**.** *The advantage $\mathbf{Adv}_{\Pi}^{\mathsf{uniqueness}}(\mathcal{A})$ of adversary $\mathcal{A}$ against the uniqueness of Π is the probability that, when $\mathcal{A}$ interacts with protocol Π, there are two sessions π, π' such that $\pi.uid = \pi'.uid$ and $\pi.role = \pi'.role$.*

For example, in TLS, a suitable *uid* is the pair of nonces sent by the client and server in their *hello* messages. To guarantee partnering upon completion, a protocol needs to protect the messages that influence the *uid* against man-in-the-middle attacks. As some configurations—in particular those where entity authentication is optional—do not protect against man-in-the-middle attacks, our definition depends on a predicate *PS* that indicates configurations that provide **P**artnering **S**ecurity. Typically, these configurations demand peer authentication.

Definition 3 (Partnering security)**.** *The advantage $\mathbf{Adv}_{\Pi, PS}^{\mathsf{partnering}}(\mathcal{A})$ of adversary $\mathcal{A}$ against the partnering security of Π is the probability that, when $\mathcal{A}$ interacts with protocol Π, there is an unpartnered session π such that $\pi.complete = \mathsf{true}$ and $PS(\pi.cfg)$ holds.*

That is, there are *at least* two sessions that assign the same value to the session variable *uid*.

2.2 Multi-mode authentication

We now define authentication for protocols in which long-term identifiers (e.g. public keys or pre-shared key identifiers) of peers are themselves negotiated. This is also known as the *post-specified peer setting* [20] and the type of authentication (e.g. mutual or bilateral) is determined as part of the negotiation [25]. We incorporate entity identifiers eid_r for $r \in \{I, R\}$ and authentication type, together with the negotiated cryptographic algorithms in the *mode* variable. We write eid_r as shorthand for $mode.eid_r$. As algorithms can be weak, keys can be compromised, and authentication can be unilateral, whether participants get guarantees depends crucially on the outcome of negotiation.

While the predicate *PS* for partnering is defined over configurations fixed upon creation of a session, our authentication definition depends on a predicate $Auth(mode, r)$, which holds when *mode* is expected to authenticate role $\bar{r}$. Authentication as defined by Lowe [39] guarantees agreement on the variables of authenticated peers. Lowe also defines two additional properties *aliveness* and *weak agreement* out of which *agreement* is the strongest. We now define agreement and a weaker notion of it, called pre-agreement. In Lowe's hierarchy, pre-agreement falls between aliveness and agreement, but is incomparable to weak agreement. We consider authentication with agreement. Authentication with pre-agreement can be defined analogously.

Jumping ahead, we will see that entity authentication and downgrade security are related, and the difference between agreement and pre-agreement will help us to understand the relationship between entity authentication and downgrade security. We discuss this in more detail in Section 3.2.

Definition 4 (Agreement). *A session π agrees with π' on x when $\pi.x = \pi'.x$. For agreement on a set X we require that π agrees with π' on all $x \in X$.*

Definition 5 (Pre-agreement). *A session π pre-agrees with π' on x when $\pi.x = \pi'.x$ or $\pi'.x = \bot$. For agreement on a set X we require that π pre-agrees with π' on all $x \in X$.*

Definition 6 (Multi-mode authentication). *A session π completes maliciously for X when $\pi.complete =$* true *but there is no partnered session π' matching π that agrees with π on X.*

The advantage $\mathbf{Adv}^{\mathsf{mm\text{-}auth}}_{\Pi, Auth, X}(\mathcal{A})$ of an adversary $\mathcal{A}$ against the multi-mode authentication security with agreement on X of protocol Π is the probability that, when $\mathcal{A}$ interacts with protocol Π, a session π completes maliciously for X and $Auth(\pi.mode, \pi.role)$ holds.

Let $r = \pi.role$. Note that $Auth(\pi.mode, r)$ typically includes the requirement that the long term key $\pi.eid_{\bar{r}}$ of the peer is honest. If, as in SIGMA-N, the mode is secure against *key-compromise impersonation* attacks [32] then $\pi.eid_r$ need not be honest. In addition, the predicate *Auth* models concurrent mixed-mode authentication. A protocol *mode* provides mutual-authentication if $Auth(\pi.mode, r)$ holds regardless of r. It provides server-only authentication if only $Auth(\pi.mode, I)$ holds, i.e., only clients get guarantees.

Observe that the authentication mode is itself negotiated. The same long-term keys eid_r routinely appear in different modes and protocols may assign the same *key* in different modes. Agreement on *mode* and other variables may be critical for higher-level protocols; *mode* may include record algorithms and using the same keys with different algorithms may lead to agile security problems. In any case it contains the entity identifiers that should be in agreement to avoid identity confusion attacks [23]. As we will see, protocols need to have sufficient downgrade resilience to guarantee that the preferred authentication mode is negotiated.

2.3 Key-indistinguishability and user privacy

Classical definitions of key indistinguishability are parameterized by a freshness predicate *Fresh* that determines the sessions with uncompromised keys. Key indistinguishability requires that for fresh sessions, an adversary cannot tell apart the real session key from a random one.

Definition 7 (Key indistinguishability). *The key indistinguishability experiment begins by sampling a bit b at random. An adversary $\mathcal{A}$ interacts with the protocol Π via the usual oracles and an additional oracle* Test *that may be called at most once. A query* Test(π) *sets k_0 to a randomly sampled key, k_1 to $\pi.key$ (or a randomly sampled key if $\pi.key = \bot$), and returns k_b.*

Let p be the probability that at the end of the experiment, adversary $\mathcal{A}$ outputs b and $Fresh(\pi)$ holds for the test session π, if any. The key indistinguishability advantage of $\mathcal{A}$ is $\mathbf{Adv}^{\mathsf{key\text{-}ind}}_{\Pi, Fresh}(\mathcal{A}) \triangleq |2p - 1|$.

For SIGMA-N, a suitable *Fresh* predicate holds for π when the group in $\pi.mode$ is strong, $\mathcal{A}$ neither queried Reveal(π) nor Reveal(π') for a matching session π', and $\pi.eid_{\bar{r}}$ is honest.

Identity protection and *deniability* are other orthogonal security requirements of key-exchange protocols. We do not formally capture them in this work, but note that many design decisions in real-world key-exchange protocols are motivated by user privacy in addition to the standard security goals key indistinguishability and entity authentication.

2.4 Instantiating our model for SIGMA-N

Consider the SIGMA-N protocol of Fig. 1. The configurations should include sufficient detail to determine the negotiated mode. We thus include the acceptable *groups* and a function *PK* from identities to peer public keys. The latter would normally be implemented by looking up the public key of the peer in a certificate store. We thus have variables

$$\begin{aligned} \mathit{cfg} &\triangleq \begin{cases} (I, A, pk_A, PK, \mathit{groups}) & \text{for initiator } I \\ (R, B, pk_B, PK, \mathit{groups}) & \text{for responder } R \end{cases} \\ \mathit{uid} &\triangleq (g^x, g^y) \\ \mathit{mode} &\triangleq (G_i, pk_A, pk_B)\ . \end{aligned}$$

3 Defining Downgrade Resilience

Downgrade resilience is motivated by protocols such as SIGMA-N that despite satisfying all of the definitions above remain vulnerable to practical attacks. We model the desired outcome of negotiation using a function *Nego* that maps two configurations with opposite roles picked from the universe of all supported configurations $\mathcal{C}$ to the protocol mode negotiated (if any) in the absence of active adversaries. Formally, if a session π talking to a session π' completes, it must be the case that $\pi.\mathit{mode} = \mathit{Nego}_r(\pi.\mathit{cfg}, \pi'.\mathit{cfg})$, where Nego_r is an abbreviation defined by case:

$$\mathit{Nego}_r(\mathit{cfg}_r, \mathit{cfg}_{\bar{r}}) \triangleq \begin{cases} \mathit{Nego}(\mathit{cfg}_r, \mathit{cfg}_{\bar{r}}) & \text{when } r = I \\ \mathit{Nego}(\mathit{cfg}_{\bar{r}}, \mathit{cfg}_r) & \text{when } r = R\ . \end{cases}$$

Definition 8 (Negotiation correctness)**.** *The protocol negotiation is correct if, whenever a session π with role r and configuration $\mathit{cfg}_r \in \mathcal{C}$ completes, there exists a peer configuration $\mathit{cfg}_{\bar{r}} \in \mathcal{C}$ such that $\pi.\mathit{mode} = \mathit{Nego}_r(\mathit{cfg}_r, \mathit{cfg}_{\bar{r}})$.*

This property captures that, if a protocol mode is disabled in a configuration, then it cannot be negotiated. Although we expect this basic property to hold unconditionally, implementation errors may break it. For instance, the FREAK attack stems from TLS clients that do not offer export ciphersuites but still accept export-grade RSA keys. An implementation of SIGMA-N in which an initiator accepts groups it did not propose would also fail to satisfy negotiation correctness.

Downgrade security complements negotiation correctness. Informally, a protocol is *downgrade secure* when two sessions of opposite roles with the same unique identifier *uid* always negotiate the mode *prescribed* by their configurations. Hence, downgrade security concerns situations in which *one participant can save the other*, even if the latter supports broken cryptography. However, we have to assume that at least some of the mechanisms of the protocol (e.g., its signature modes) are strong enough. Conversely, if both participants enable (among others) a mode that is entirely insecure, then there is no cryptographically sound way to prevent an attacker from downgrading their connection.

Our definition is parameterized by a downgrade-protection predicate *DP* on pairs of configurations.

$\mathit{DP}(\mathit{cfg}_r, \mathit{cfg}_{\bar{r}})$ indicates the pairs of configurations from which we expect downgrade protection; it is not necessarily symmetric. By convention, cfg_r is the local configuration, $\mathit{cfg}_{\bar{r}}$ is the peer configuration, and when $\mathit{DP}(\mathit{cfg}_r, \mathit{cfg}_{\bar{r}})$ holds, we expect that the local session is protected.

Definition 9 (Downgrade security)**.** *A session π is downgraded when $\pi.complete = \mathsf{true}$ and there is a partnered session π' such that $DP(\pi.cfg, \pi'.cfg)$, and $\pi.mode \neq Nego_{\pi.role}(\pi.cfg, \pi'.cfg)$.*

The advantage $\mathbf{Adv}^{\mathsf{downgrade}}_{\Pi, DP, X}(\mathcal{A})$ of $\mathcal{A}$ against downgrade security with pre-agreement on X is the probability that, when $\mathcal{A}$ terminates after interacting with Π, there exists a session π that either is downgraded or does not pre-agree with a partnered session π' on X. We write $\mathbf{Adv}^{\mathsf{downgrade}}_{\Pi, DP}(\mathcal{A})$ when $X = \{\}$.

Note also that only partnered sessions get downgrade protection guarantees, so our definition is meaningful only for protocols for which partnering security holds. Thus, for role r, if $DP(cfg_r, cfg_{\bar{r}})$ holds for any peer configuration $cfg_{\bar{r}}$, we also need that $PS(cfg_r)$ holds; we write this concisely as $DP \subseteq_r PS$, and observe that this property holds in our case studies. Downgrade security is a complementary, intuitively stronger property.

The DP predicate for downgrade protection plays a role similar to $Auth$ for authentication, but it depends only on static configurations and on the honesty of long-term credentials. This reflects that downgrade protection should depend only on the inputs to the negotiation, and not the negotiation itself, which may be under the influence of an adversary.

Our formal configurations are session-specific, and do not necessarily coincide with concrete configuration in real-world protocol deployments. In particular, each configuration contains credentials only for the intended peer (e.g. cached certificates, key fingerprints). As an example, our configurations for TLS include the authentication settings of the session: the client's configuration expresses its intent to communicate with a particular server, who may support multiple negotiable certificates, e.g., those negotiated using the server name indication (SNI) extension [17]. SNI enables servers to host multiple 'virtual' servers at a single underlying network address.

Ideally, $DP(\pi.cfg, \cdot)$ would hold regardless of the second configuration. Anticipating on our results, this is the case for SSH, where DP is defined as follows: the configuration of π must require authentication of its peer, all peer keys accepted by π must be honest, and all signature algorithms must be (agile) strong. However, this is not the case e.g. for TLS 1.2 clients, which do not get downgrade protection with servers that support weak Diffie-Hellman groups.

Generalizing downgrade security For simplicity, our definitions above assume there is at most one correct mode reachable from configurations $\pi.cfg$ and $\pi'.cfg$, and consider adversaries that lead sessions to pick any other mode.

More generally, we may let *Nego* return a set of equally-acceptable modes and tolerate attacks that influence which of these modes is picked by partnered sessions. In that case, of course, we would still insist that the two sessions agree on the negotiated mode.

As an example, we may interpret configurations as sets of acceptable modes, and let *Nego* compute their intersection. As long as the protocol is negotiation-correct and guarantees agreement on *mode*, it would be downgrade secure for this generalized definition.

3.1 Downgrade resilience of SIGMA-N

SIGMA-N configurations are tuples $(r, ID, pk_{ID}, PK, groups)$ where PK is a function mapping identities to public keys. The negotiation function describes the correct mode upon completion. Given a function *nego* that selects the preferred common group, *Nego* is defined as

$$Nego(cfg_I, cfg_R) \triangleq \begin{cases} (nego(cfg_I.groups, cfg_R.groups), cfg_I.pk_A, cfg_R.pk_B) & \text{if } cfg_I.pk_A = cfg_R.PK(A) \\ \bot & \text{otherwise .} \end{cases}$$

For such a *Nego* function, *DP* can hold only for pairs of configurations with at most one group in common. This can be seen from an investigation of the attack in Fig. 2.

However, observe that if *DP* guarantees the honesty of peer public keys, a participant only accepts groups whitelisted by its partnered peer. Thus, SIGMA-N is downgrade secure if we generalize *Nego* to tolerate the negotiation of any mode that uses a group in the intersection of the groups whitelisted by both participants, i.e.

$$\mathit{Nego}(\mathit{cfg}_I, \mathit{cfg}_R) \triangleq \begin{cases} \{(G, \mathit{cfg}_I.\mathit{pk}_A, \mathit{cfg}_R.\mathit{pk}_B) \mid G \in \mathit{cfg}_I.\mathit{groups} \cap \mathit{cfg}_R.\mathit{groups}\} & \text{if } \mathit{cfg}_I.\mathit{pk}_A = \mathit{cfg}_R.\mathit{PK}(A) \\ \varnothing & \text{otherwise .} \end{cases}$$

3.2 Agreement vs. pre-agreement.

Using SIGMA-N as an example, we now discuss the role that agreement plays in the definition of multi-mode authentication and the analogous role that pre-agreement plays in the definition of downgrade security and explain why we chose to define downgrade security with respect to pre-agreement.

The reason is that agreement on *mode* (or some of its parts) is desirable but we do not consider it essential for downgrade protection. A downgrade attack means that one or both partnered sessions π and π' assign a weaker mode than the prescribed one. In particular, if the session π' does not assign a mode, then it has not been downgraded. Pre-agreement assures that if the partner session π' of π assigns some mode, then this mode agrees with the mode of π. Conversely, for configurations cfg_r and $\mathit{cfg}_{\bar{r}}$ for which both $\mathit{DP}(\mathit{cfg}_r, \mathit{cfg}_{\bar{r}})$ and $\mathit{DP}(\mathit{cfg}_{\bar{r}}, \mathit{cfg}_r)$ hold, we have downgrade protection with pre-agreement on *mode*.

Interestingly, considering pre-agreement instead of agreement sometimes allows us to shave off a round of communication from a sub-protocol, and in a sense, the same phenomenon can be observed in the context of entity authentication. Let us contemplate the agreement vs pre-agreement issue more closely in the context of SIGMA-N.

SIGMA-N inherits the security properties of SIGMA [36] which explicitly does not guarantee *peer awareness*. When A completes a session with peer B, A has no guarantee that B has initiated the corresponding session with peer A. It thus only achieves authentication with pre-agreement. More precisely, authentication with pre-agreement and key-indistinguishability, respectively, hold for *Auth* and *Fresh* predicates that both require strong groups and honest keys. Since g^x and g^y are signed it also provides partnering security, assuming that the signatures scheme is (agile) secure, even if the group is weak.

A straightforward patch to address the downgrade attack on SIGMA-N would be to have the initiator signs its proposal and the responder signs the proposal and its choice. This would indeed guarantee downgrade security (with pre-agreement on *mode*), which is good.

In turn, it does neither provide peer awareness (a.k.a. authentication with agreement) nor downgrade security with *agreement* on *mode*. The reason is that the responder completes first without receiving any confirmation from the initiator, i.e., *Nego* is defined even when $\mathit{cfg}_R.\mathit{pk}_B \neq \mathit{cfg}_I.\mathit{PK}(B)$. This is fine from the perspective of downgrade security. Whether it is also appropriate with respect to entity authentication is subject to preferences/design choices in the area of entity authentication that are quite independent of downgrade security.

3.3 Downgrade resilience and multi-mode security

Protocol analysts often consider protocols restricted to specific modes and configurations. For instance it is common practice to analyze individual protocol modes in isolation. Similarly we can restrict the initial universe of possible configurations of a protocol to those that provide downgrade protection. Consider sets of configurations C_I and C_R picked by initiators and responders respectively. We consider restricted protocols in which sessions abort whenever they are initialized with a configuration outside of the set $C_I \cup C_R$.

Definition 10 (Protected configurations)**.** *Let DP be a downgrade protection predicate. A pair of sets of configurations* $(C_r, C_{\bar{r}})$ *gives downgrade protection to role* r *if* $C_r \times C_{\bar{r}} \subseteq DP$.

The following theorem expresses that when downgrade security holds, only the security of modes that can be negotiated in the absence of an adversary matters. That is, if peers support insecure modes, but with such a low priority that they never negotiate them on their own, then these modes do not affect security in the presence of an adversary.

Theorem 1 (Downgrade resilience and multi-mode authentication)**.** *Let* Π *be a protocol,* $(C_r, C_{\bar{r}})$ *sets of configurations, DP a downgrade protection predicate, and* $\mathcal{N} = \{Nego_r(cfg_r, cfg_{\bar{r}}) \mid cfg_r, cfg_{\bar{r}} \in C_r \times C_{\bar{r}}\}$ *the modes negotiable without adversary influence. If* $DP \subseteq_r PS$ *and*

- $(C_r, C_{\bar{r}})$ *gives downgrade protection to* r*,*
- Π *is multi-mode authentication secure for* $Auth, X$*,*
- Π *is partnering secure for PS, and*
- Π *is downgrade secure for DP,*

then the protocol Π *restricted to configurations in* $C_r \cup C_{\bar{r}}$ *is multi-mode authentication secure for a more lax* $Auth'$ *predicate that deems all modes outside of* $\mathcal{N}$ *as "good", i.e.* $Auth'(m, role) \triangleq Auth(m, role) \vee (m \notin \mathcal{N} \wedge role = r)$*. Concretely, given an adversary* $\mathcal{A}$ *against authentication for* $Auth', X$*, we have*

$$\mathbf{Adv}^{\mathit{auth}}_{\Pi', Auth', X}(\mathcal{A}) \leq \mathbf{Adv}^{\mathit{partnering}}_{\Pi, PS}(\mathcal{A}) + \mathbf{Adv}^{\mathit{downgrade}}_{\Pi, DP}(\mathcal{A}) + \mathbf{Adv}^{\mathit{auth}}_{\Pi, Auth, X}(\mathcal{A}) ,$$

where Π' *is* Π *restricted to configurations* $C_r \cup C_{\bar{r}}$*.*

Proof sketch. Consider the multi-mode authentication experiment G_0 for Π'. Let S hold when at some point through G_0 a session π completes maliciously on X and $Auth(\pi.mode, \pi.role)$ holds (i.e., $\mathcal{A}$ succeeds in breaking authentication iff S holds at the end of G_0).

Game G_1 behaves as G_0 except it aborts just before a session π of role r would complete without being partnered. Because of the restriction in Π' and the hypothesis that $DP \subseteq_r PS$, it must be the case that $PS(\pi.cfg)$. Thus, any time G_1 aborts, $\mathcal{A}$ succeeds in breaking the partnering security of Π', and thus that of Π. Hence, the difference in the probability of S between G_0 and G_1 is at most $\mathbf{Adv}^{\mathsf{partnering}}_{\Pi, PS}(\mathcal{A})$.

Game G_2 behaves as G_1 except it aborts just before a session π of role r would complete and there is a partnered session π' such that $\pi.mode \neq Nego_r(\pi.cfg, \pi'.cfg)$. The difference in the probability of S between G_1 and G_2 is at most $\mathbf{Adv}^{\mathsf{downgrade}}_{\Pi, DP}(\mathcal{A})$ since any time G_2 aborts but G_1 does not, $\mathcal{A}$ succeeds in breaking the downgrade security of Π', and thus that of Π.

By definition of $\mathcal{N}$, G_2 never completes with a session of role r assigning a mode outside of $\mathcal{N}$. Consequently, the probability of S in this game is at most

$$\mathbf{Adv}^{\mathsf{auth}}_{\Pi', Auth, X}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{auth}}_{\Pi, Auth, X}(\mathcal{A}) . \quad \square$$

Interestingly, partnering security is similar to the aliveness requirement in some (single-mode) security definitions which Krawczyk [36] does not consider as fundamental for key-exchange security. Our second game transformation however only works if a partnered session π' with the same *uid* exists. Otherwise an abort in G_2 cannot be translated into a downgrade security attack.

For key-indistinguishability and a freshness predicate $\mathit{Fresh}(\pi)$ that requires for matching π' with $\pi'.\mathit{mode} \neq \bot$ that $\pi.\mathit{mode} = \pi'.\mathit{mode} = \mathit{Nego}_r(\pi.\mathit{cfg}, \pi'.\mathit{cfg})$, we have an analogous theorem for

$$\mathit{Fresh}'(\pi) \triangleq \mathit{Fresh}(\pi) \vee (\pi.\mathit{mode} \notin \mathcal{N} \wedge \pi.\mathit{role} = r) \; .$$

In the context of TLS, Bhargavan et al. [15] observe that this is a sufficient condition for the security of session keys that are released before the handshake completes.

3.4 Downgrade secure sub-protocols

We are interested in minimal core sub-protocols that guarantee downgrade security. To justify our use of sub-protocols in further sections as a sound abstraction of the full protocol we use simulation. Our sub-protocols can take additional input as part of Init and Send queries to allow for an accurate simulation of the execution of the full protocol including each party's local state. This is akin to the partially specified protocols of Rogaway and Stegers [49] which allow for adversarially defined protocol details and the sub-protocols of Bergsma et al. [11] which allow for additional signing oracles (restricted to not breaking security of the sub-protocol).

For simplicity, the following definition leaves out details about handling of long-term keys and corruption models. When filling in the details for a particular setting, we require the simulation to be accurate with respect to e.g. corruption, so that it issues exactly the same corruption queries as in the full protocol. We model access to session variables using oracles that just return the value of the corresponding variable.

Definition 11 (Sub-protocol)**.** *A protocol $\widetilde{\Pi}$ is a sub-protocol of Π for X if we have an efficient simulator $\mathcal{S}$ with only oracle access to $\widetilde{\Pi}$ that is accurate and indistinguishable in the following sense:*

(a) While $\mathcal{S}$ can simulate most oracles of Π whatever way it wants, it has to accurately forward requests and replies to corruption oracle as well as oracles giving access to variables in X to the same oracles in $\widetilde{\Pi}$.

(b) We require that $\mathcal{S} \circ \widetilde{\Pi}$ is indistinguishable from Π for an information-theoretic distinguisher.

Formally, we model a protocol (and a simulator) as a collection of oracles sharing state, each oracle being a probabilistic algorithm. The composition $\mathcal{S} \circ \widetilde{\Pi}$ of a simulator $\mathcal{S}$ and a sub-protocol $\widetilde{\Pi}$ is well-defined when $\widetilde{\Pi}$ includes all algorithms called by the oracles of $\mathcal{S}$. The composition itself is a new collection of algorithms, one for each oracle of $\mathcal{S}$. Operationally, the oracles of $\mathcal{S} \circ \widetilde{\Pi}$ behave as the algorithmic composition of the oracles of $\mathcal{S}$ and $\widetilde{\Pi}$—intuitively $\mathcal{S}$ uses the oracles of $\widetilde{\Pi}$ as subroutines. Similarly, we model an adversary $\mathcal{A}$ as a single probabilistic algorithm with access to oracles, and the composition $\mathcal{A} \circ \mathcal{S}$ (resp. $\mathcal{A} \circ \Pi$) behaves as the algorithmic composition of this algorithm with the oracles of $\mathcal{S}$ (resp. Π).

As the next theorem shows, simulation allows to lift security properties satisfied by a sub-protocol to the full protocol.

Theorem 2 (Downgrade security lifting)**.** *Let $\widetilde{\Pi}$ be a sub-protocol of a protocol Π for session variables $\{\mathit{cfg}, \mathit{uid}, \mathit{mode}, \mathit{key}, \mathit{complete}\} \cup X$, and DP a downgrade protection predicate. Let $\mathcal{S}$ be a simulator for $\widetilde{\Pi}$ as in Definition 11. Then, for any adversary $\mathcal{A}$ against the DP-downgrade security of Π with agreement on X, $\mathcal{A} \circ \mathcal{S}$ is an adversary against the downgrade security of $\widetilde{\Pi}$ with agreement on X, and*

$$\mathbf{Adv}^{\mathsf{downgrade}}_{\widetilde{\Pi}, DP, X}(\mathcal{A} \circ \mathcal{S}) = \mathbf{Adv}^{\mathsf{downgrade}}_{\Pi, DP, X}(\mathcal{A}) \; .$$

Proof sketch. If $\mathcal{A}$ is successful when interacting with Π through the protocol oracles, then during the downgrade security experiment there must be a session π partnered with a session π' such that $DP(\pi.cfg, \pi'.cfg)$ holds and either $\pi.mode \neq Nego_{\pi.role}(\pi.cfg, \pi'.cfg)$ or π and π' disagree on X. Let E denote this event. Note that the probability of E in the experiment $\mathcal{A} \circ \Pi$ is exactly $\mathbf{Adv}^{\mathsf{downgrade}}_{\Pi, DP, X}(\mathcal{A})$.

Now, since the simulation $\mathcal{S} \circ \widetilde{\Pi}$ is accurate with respect to all variables this event depends on, and $\mathcal{S} \circ \widetilde{\Pi}$ is indistinguishable from Π for $\mathcal{A}$, the probability of E ocurring in the experiment $\mathcal{A} \circ (\mathcal{S} \circ \widetilde{\Pi})$ is the same as in the experiment $\mathcal{A} \circ \Pi$.

Because the composition operator $\circ$ is such that $\mathcal{A} \circ (\mathcal{S} \circ \widetilde{\Pi}) = (\mathcal{A} \circ \mathcal{S}) \circ \widetilde{\Pi}$, we conclude by construing the composition of $\mathcal{A}$ and $\mathcal{S}$ as an adversary against the downgrade security of $\widetilde{\Pi}$ with pre-agreement on X. $\square$

An analogous theorem holds for partnering security.

3.5 Downgrade security by whitelisting

Consider a protocol that is negotiation correct and guarantees multi-mode authentication with agreement on all variables that influence the computation of *mode*, then we get downgrade protection for

$$DP(cfg, .) \triangleq \forall cfg'.\ Auth(Nego_{cfg.role}(cfg, cfg'), cfg.role)\,.$$

That is, all negotiable modes from downgrade secure configurations must provide authentication security. This generalizes the *Negotiation-authentication* theorem of [24].

Theorem 3 (Negotiation and authentication)**.** *Let Π be a protocol, Auth a multi-mode authentication predicate. If Π is multi-mode authentication secure for Auth, X, then the protocol Π is downgrade secure for $DP(cfg, .) \triangleq \forall cfg'.\ Auth(Nego_{cfg.role}(cfg, cfg'), cfg.role)$ with (pre-)agreement on X. Concretely, given an adversary $\mathcal{A}$ against downgrade security for DP, X, we have*

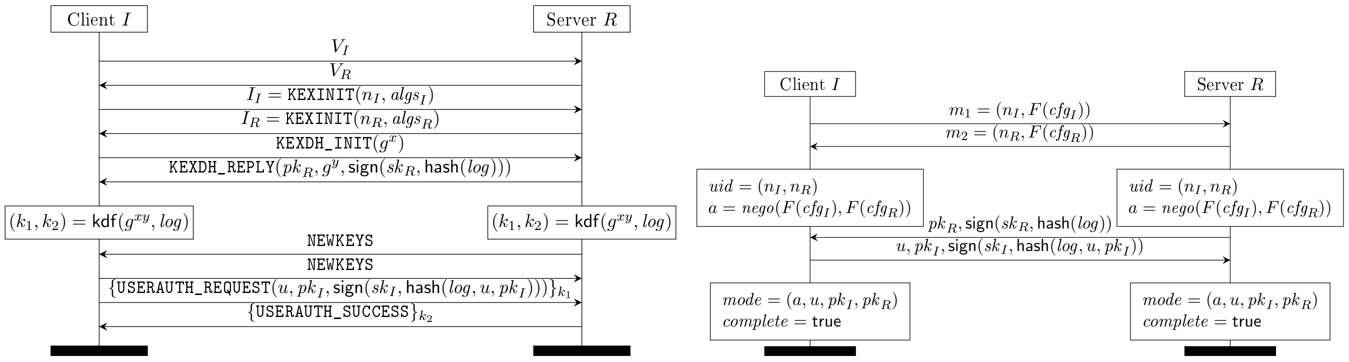
$$\mathbf{Adv}^{\mathit{downgrade}}_{\Pi, DP, X}(\mathcal{A}) \leq \mathbf{Adv}^{\mathit{auth}}_{\Pi, Auth, X}(\mathcal{A})\,.$$

4 Secure SHell

Figure 3a models a run of the SSHv2 [54] protocol with a client that authenticates using the `publickey` method [53]. We analyze the downgrade security of this protocol using the sub-protocol shown on Figure 3b. The functions H, H' in these figures stand for the composition of a fixed injective formatting function and a negotiated hash function. Note that there are potential downgrade attacks in SSHv2 from `publickey` authentication to other mechanisms like `password` but the protocol we consider does not model the negotiation of the authentication mechanism. We stress that our analysis only applies assuming servers are configured to require public key authentication.

We next describe the notation used in Figure 3 and the way the protocol works. We then prove that *SSH-sub* is indeed a sub-protocol of the full protocol for the variables they have in common.

Client and server configurations include lists *algs* of key exchange, server signature, encryption and MAC algorithms ordered by preference. We let $F(cfg) = cfg.algs$. Each party computes the negotiated ciphersuite independently, following the rules in the protocol specification [54, Sect. 7.1], which we encode in a *nego* function. Roughly, these rules dictate that the first algorithm for each category in cfg_I that is also in cfg_R be selected. Each session locally assigns $nego(F(cfg_I), F(cfg_R))$ to a. In addition, a client



Client I
Server R
V_I
V_R
$I_I = \texttt{KEXINIT}(n_I, algs_I)$
$I_R = \texttt{KEXINIT}(n_R, algs_R)$
$\texttt{KEXDH\_INIT}(g^x)$
$\texttt{KEXDH\_REPLY}(pk_R, g^y, \mathsf{sign}(sk_R, \mathsf{hash}(log)))$
$(k_1, k_2) = \mathsf{kdf}(g^{xy}, log)$
$(k_1, k_2) = \mathsf{kdf}(g^{xy}, log)$
NEWKEYS
NEWKEYS
$\{\texttt{USERAUTH\_REQUEST}(u, pk_I, \mathsf{sign}(sk_I, \mathsf{hash}(log, u, pk_I)))\}_{k_1}$
$\{\texttt{USERAUTH\_SUCCESS}\}_{k_2}$
Client I
Server R
$m_1 = (n_I, F(cfg_I))$
$m_2 = (n_R, F(cfg_R))$
$uid = (n_I, n_R)$
$a = nego(F(cfg_I), F(cfg_R))$
$uid = (n_I, n_R)$
$a = nego(F(cfg_I), F(cfg_R))$
$pk_R, \mathsf{sign}(sk_R, \mathsf{hash}(log))$
$u, pk_I, \mathsf{sign}(sk_I, \mathsf{hash}(log, u, pk_I))$
$mode = (a, u, pk_I, pk_R)$
$complete = \mathsf{true}$
$mode = (a, u, pk_I, pk_R)$
$complete = \mathsf{true}$

(a) $log = H(V_I, V_R, I_I, I_R, pk_R, g^x, g^y, g^{xy})$ (b) $log = H'(m_1, m_2, pk_R, -)$

Figure 3: SSHv2 mutually-authenticated key exchange: (a) full protocol and (b) sub-protocol *SSH-sub*.

configuration cfg_I includes a user name and a service name u, a function PK_I mapping a pair (a, u) to a public key, and a function PKs_R mapping a value a to a set of acceptable server public keys. Conversely, a server configuration cfg_R includes a function PK_R mapping a value a to a public key, and a function PKs_I mapping a pair (a, u) to a set of acceptable client public keys. For instance, in OpenSSH the keys $cfg_I.PKs_R$ of acceptable server public keys are taken from the clients `known_hosts` file, whereas the keys $cfg_R.PKs_I$ of acceptable client public keys are taken from the `.ssh/authorized_keys` file in the home directory of the user on the server.

In terms of the template in Section 2, the sub-protocol uses the following session variables:

$$\begin{aligned} cfg &\triangleq \begin{cases} (I, algs, u, PK_I, PKs_R) & \text{for } I \\ (R, algs, PK_R, PKs_I) & \text{for } R \end{cases} \\ uid &\triangleq (n_I, n_R) \\ mode &\triangleq (a, u, pk_I, pk_R)\ . \end{aligned}$$

Client and server exchange nonces and their algorithmic preferences $F(cfg_I), F(cfg_R)$. The server then selects a compatible signature key pair (pk_R, sk_R) and signs a hash log that includes the first two exchanged messages. When receiving this message, the client checks that pk_R is an acceptable server key in its local configuration, computes log locally and verifies the server signature. If the signature verifies, it selects a key pair (pk_I, sk_I) in its configuration for authenticating and sends back to the server a signature over log, u, and pk_I. When receiving this message, the server checks that pk_I is an acceptable client key in $cfg_R.PKs_I(a, u)$. Each party completes the session upon successfully verifying the peer signature, otherwise aborts. Formally, a client aborts if $pk_R \notin cfg_I.PKs_R(a)$; otherwise it assigns the following value to $mode$:

$$(nego(F(cfg_I), F(cfg_R)),\ cfg_I.u,\ cfg_I.PK_I(a, u),\ pk_R)$$

The server's behavior is specified analogously.

We augment the Send oracles of each a session in the sub-protocol with extra parameters that allow to fill in the blank $(-)$ used to compute log. This allows a simulator to compute signatures on the same values as the full protocol, as needed to consistently answer Send queries. Consequently, we allow an adversary against the downgrade security of the sub-protocol to fill in $-$ parameters arbitrarily.

We complete our security model with oracles $pk \leftarrow \mathsf{KeyGen}$ for key generation, $sk \leftarrow \mathsf{Corrupt}(pk)$ for adaptive corruption, and $\mathsf{Coerce}(pk)$ for adversarial key registration. A public key pk is honest if it was generated by a query to oracle KeyGen but not corrupted by a Corrupt query.

Theorem 4 (Simulation)**.** *SSH-sub (Fig. 3b) is a sub-protocol of SSH (Fig. 3a) for their common variables.*

Proof sketch. The sub-protocol is oblivious of the Diffie-Hellman exchange in the full protocol, so the simulator generates fresh Diffie-Hellman shares of his own for each session. When needed, the simulator $\mathcal{S}$ forwards queries to *SSH-sub* after applying message parsing and formatting functions. To simulate signatures of honest sessions $\mathcal{S}$ uses the Diffie-Hellman shares it has computed and the messages it has received to fill in the value of the extra parameter $-$ of oracles of the sub-protocol. Note that the adversary knows the secret exponents of an instance's Diffie-Hellman shares and so it can always compute the encryption keys k_1, k_2 needed to simulate the last two messages of the full protocol. □

4.1 SSHv2 is partnering and downgrade secure

A remarkable property of the downgrade protection sub-protocol of mutually-authenticated SSHv2 is that, because both client and server sign (a hash of) the inputs to the *nego* function, downgrade protection security relies only on the honesty of the signature keys, the collision resistance of the hash algorithm, and the strength of the signature algorithms. Notably, it does not rely on the key exchange algorithm being strong or contributive, not even on it providing high entropy inputs to H. This means that we can prove this protocol secure for a predicate DP that only constrains the signature and hash algorithms of cfg_r, and requires honesty of peer public keys in $cfg_r.PKs_{\bar{r}}$, but has no requirements on $cfg_{\bar{r}}$.

We prove the partnering and downgrade security of SSHv2 with `publickey` client authentication under the agile security assumptions on hash functions and signatures that we present next.

4.1.1 Agile hash functions and signatures

As protocol participants may negotiate different hash functions we need to capture collisions across hash functions.

Definition 12 (Agile collision resistance)**.** *Let $h^\star$ be a hash function, and $\mathcal{H}$ a set of hash functions. Consider the game:*

- $h, v, v' \leftarrow \mathcal{A}()$
- *Return* $h^\star(v) = h(v') \wedge v \neq v'$

The collision resistance advantage of $\mathcal{A}$, $\mathbf{Adv}^{\mathsf{CR}}_{h^\star, P}(\mathcal{A})$ is the probability that the game returns true.

If the ranges of hash functions are disjoint, agile collision resistance reduces to ordinary collision resistance. Bhargavan et al. [15] also define existential unforgeability under chosen-message attacks (EUF-CMA) for agile hash-then-sign signatures. We here consider such signatures as primitives although typical constructions can be proved secure in the random oracle model.

Definition 13 (Agile EUF-CMA security)**.** *Consider an agile signature scheme $s = (\mathsf{keygen}, \mathsf{sign}, \mathsf{verify})$. Let $p^\star$ be an agility parameter, and $\mathcal{P}$ a set of parameters. Consider the forgery game:*

- *Let* $pk, sk \leftarrow \mathsf{keygen}()$

- *Set $M := \{\}$ and run $m, \sigma \leftarrow \mathcal{A}^{\mathsf{Sign}}(pk)$*

- *Return $m \notin M \wedge \mathsf{verify}(pk, p^\star, m, \sigma)$*

where $\mathsf{Sign}(p, m)$ *returns* $\perp$ *if* $p \notin \mathcal{P}$ *and otherwise sets* $M := M \cup \{m\}$ *before returning* $\mathsf{sign}(sk, p, m)$.

The advantage $\mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}_{s, p^\star, \mathcal{P}}(\mathcal{A})$ *of* $\mathcal{A}$ *in forging a signature for* s *is the probability that the forgery game returns true.*

Since we proved that *SSH-sub* soundly abstracts negotiation in the full protocol, any downgrade attack on the full protocol can be turned into a downgrade attack on the sub-protocol. By virtue of Theorem 2 it suffices to prove that *SSH-sub* is downgrade secure. The same reasoning applies to partnering security.

4.1.2 Partnering security

To state and prove partnering security of SSHv2, we now define *Nego* and use the following notations. $Nego(cfg_I, cfg_R) \triangleq (a, cfg_I.u, cfg_I.PK_I(a, u), pk_R)$, where $a = nego(F(cfg_I), F(cfg_R))$ if $pk_R = cfg_R.PK_R(a)$, and $\perp$ otherwise. $\mathcal{M}^\star \triangleq \{Nego_{cfg.role}(cfg, cfg') | PS(cfg)\}$ are the modes negotiated between any pair of configurations for which the first guarantees partnering security and $\mathcal{M}$ be the set of all supported modes. $\mathcal{P}_s \triangleq \{p \mid s, p = mode.\overline{sig} \wedge mode \in \mathcal{M}\}$ are the agility parameters for the signature scheme s of a peer, $\mathcal{H}$ is the set of all supported hash algorithms, and $\mathcal{H}^\star \triangleq \{mode.hash \mid mode \in \mathcal{M}^\star\}$ are the hash algorithms used by partnering secure modes.

Theorem 5 (Partnering security of *SSH-sub*). *Let PS be such that PS(cfg) implies that all public keys in the range of* $cfg.PKs_{\overline{cfg.role}}$ *are honest. Given an adversary* $\mathcal{A}$ *against the partnering security of SSH-sub, we construct adversaries* $\mathcal{B}_{s,p,i}$ *and* $\mathcal{B}_h$ *running in about the same time as* $\mathcal{A}$ *such that* $\mathbf{Adv}^{\mathsf{partnering}}_{SSH\text{-}sub, PS}(\mathcal{A})$ *is at most*

$$\sum_{h \in \mathcal{H}^\star} \mathbf{Adv}^{\mathsf{CR}}_{h, \mathcal{H}}(\mathcal{B}_h) + \sum_{(s,p) \in \overline{sig}(\mathcal{M}^\star)} \sum_{i=1}^{n_s} \mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}_{s, p, \mathcal{P}_s}(\mathcal{B}_{s,p,i}) ,$$

where n_s *is the number of keys generated for scheme* s.

Proof sketch. Let G_0 be the original partnering security experiment. S is the event that at some point during the experiment, a session π for which $PS(\pi.cfg)$ holds completes without being partnered. Let G_1 behave like G_0, except that it aborts whenever two sessions hash messages (m_1, m_2) with different (n_I, n_R) to the same value. Using $\mathcal{A}$, we can construct for each $h \in \mathcal{H}^\star$ an adversary $\mathcal{B}_h$ such that $|\Pr_{\mathsf{G}_0}[S] - \Pr_{\mathsf{G}_1}[S]| \leq \sum_{h \in \mathcal{H}^\star} \mathbf{Adv}^{\mathsf{CR}}_{h, \mathcal{H}}(\mathcal{B}_h)$.

Game G_2 behaves as G_1, except that sessions with honest peer keys and a peer signature scheme $(s, p) \in \overline{sig}(\mathcal{M}^\star)$ abort without completing whenever they verify a signature on *log* that was not signed by another session.

To bound the difference in the probability of S between these games we use a lemma of Bhargavan et al. [16, Lemma 4]. It shows via a hybrid argument how to construct adversaries $\mathcal{B}_{s,p,i}$ that bound the probability of a forgery for an agile hash-and-sign scheme $(s, p) \in \overline{sig}(\mathcal{M}^\star)$ in an agile signature library that generates n_s honest keys for signing scheme s, thus

$$|\Pr_{\mathsf{G}_1}[S] - \Pr_{\mathsf{G}_2}[S]| \leq \sum_{(s,p) \in \overline{sig}(\mathcal{M}^\star)} \sum_{i=1}^{n_s} \mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}_{s, p, \mathcal{P}_s}(\mathcal{B}_{s,p,i}).$$

We conclude by confirming that all sessions hash (n_I, n_R) into *log* and verify a signature on *log* before completion. For those sessions for which $PS(cfg)$ holds, $pk_{\overline{cfg.role}} \in cfg.PKs_{\overline{cfg.role}}$ is honest, and we have that another session in possession of the signing key must have assigned the same $uid = (n_I, n_R)$. Finally, since the signatures computed by initiators and responders cannot be confounded, the other session must have the opposite role. □

4.1.3 Downgrade security

To prove downgrade security, we define *Nego*, $\mathcal{M}$, $\mathcal{P}_s$, and $\mathcal{H}$ as before, but re-define $\mathcal{M}^\star$, $\mathcal{H}^\star$ to use *DP* instead of *PS*, i.e. $\mathcal{M}^\star \triangleq \{Nego_{cfg.role}(cfg, cfg') \mid DS(cfg, cfg')\}$.

Theorem 6 (Downgrade security of *SSH* sub-protocol)**.** *Let DP be such that $DP(cfg,\cdot)$ implies that all public keys in the range of $cfg.PK_{\overline{cfg.role}}$ are honest. Given an adversary $\mathcal{A}$ against the downgrade security of the sub-protocol, we construct adversaries $\mathcal{B}_{s,p,i}$ and $\mathcal{B}_h$ running in about the same time as $\mathcal{A}$ such that $\mathbf{Adv}^{\mathsf{downgrade}}_{SSH\text{-}sub,\,DP}(\mathcal{A})$ is at most*

$$\frac{n^2}{2^{|uid|/2}} + \sum_{h\in\mathcal{H}^\star} \mathbf{Adv}^{CR}_{h,\mathcal{H}}(\mathcal{B}_h) + \sum_{(s,p)\in\overline{sig}(\mathcal{M}^\star)} \sum_{i=1}^{n_s} \mathbf{Adv}^{EUF\text{-}CMA}_{s,p,\mathcal{P}_s}(\mathcal{B}_{s,p,i})\,,$$

where n is the number of sessions, n_s the number of keys generated for scheme s, and $|uid|$ the size of unique identifiers.

Proof sketch. Let S be the event that there exists a downgraded session π and let G_0 be the original downgrade security experiment.

Game G_1 aborts whenever two sessions of the same role assign the same *uid*. Let n be the total number of sessions. The length of the randomness in client and server nonces is $|uid|/2$ bits. The probability that n such random values give rise to a collision is upper bounded by $n^2 2^{-|uid|/2}$. As G_0 and G_1 are equivalent up to collisions, we have $|\Pr_{\mathsf{G}_0}[S] - \Pr_{\mathsf{G}_1}[S]| \leq n^2 2^{-|uid|/2-1}$.

Game G_2 behaves as G_1, except that it aborts whenever two sessions hash messages $(m_1, m_2, -, pk_R, -)$ to the same *log*.

Using $\mathcal{A}$, we can construct for each $h \in \mathcal{H}^\star$ an adversary $\mathcal{B}_h$ such that $|\Pr_{\mathsf{G}_0}[S] - \Pr_{\mathsf{G}_1}[S]| \leq \sum_{h\in\mathcal{H}^\star} \mathbf{Adv}^{\mathsf{CR}}_{h,\mathcal{H}}(\mathcal{B}_h)$.

Observe that in G_2, every pair of nonces $(n_I, n_R) = uid$ in the message pairs (m_1, m_2) is signed at most once per role.

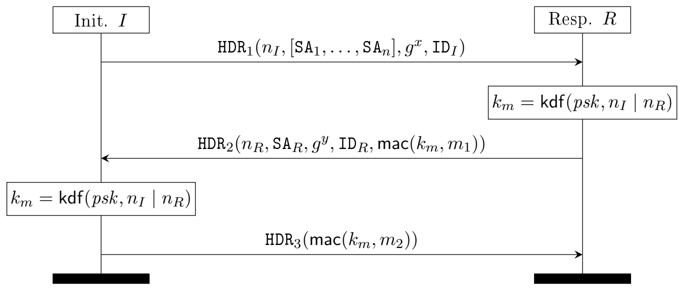
Game G_3 is the same as Game G_2, except that sessions with honest peer keys and $(s,p) \in \overline{sig}(\mathcal{M})$ abort without completing whenever they verify a signature on *log* or (log, u, pk_I) that was not signed by another session.

We again bound the difference in the success probabilities using the lemma of Bhargavan et al. [16, Lemma 4] and have

$$|\Pr_{\mathsf{G}_2}[S] - \Pr_{\mathsf{G}_3}[S]| \leq \sum_{(s,p)\in\overline{sig}(\mathcal{M}^\star)} \sum_{i=1}^{n_s} \mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}_{s,p,\mathcal{P}_s}(\mathcal{B}_{s,p,i}).$$

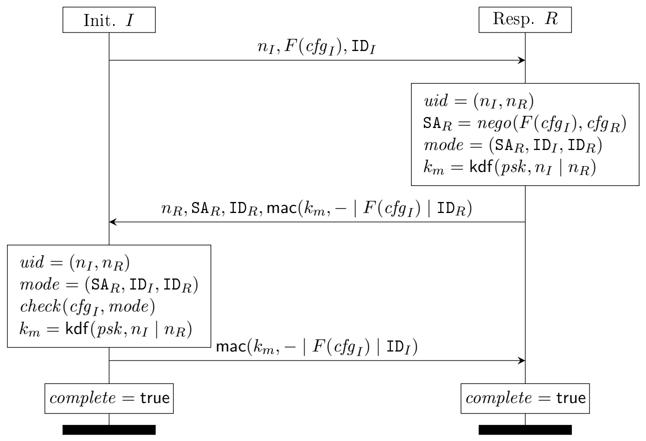
We now show that the probability that event S occurs in Game G_3 is 0. Observe that, since the signatures computed by initiators and responders cannot be confounded, in Game G_3 we never verify a message that was not signed by its unique matching peer. We now consider initiators and responders separately.

For the initiator we have that message *log* was signed by its partner, which computed it from the responders key pk_R and messages m_1 and m_2 containing the contribution of the local and the remote



Init. I
Resp. R
$\mathtt{HDR}_1(n_I, [\mathtt{SA}_1, \ldots, \mathtt{SA}_n], g^x, \mathtt{ID}_I)$
$k_m = \mathsf{kdf}(psk, n_I \mid n_R)$
$\mathtt{HDR}_2(n_R, \mathtt{SA}_R, g^y, \mathtt{ID}_R, \mathsf{mac}(k_m, m_1))$
$k_m = \mathsf{kdf}(psk, n_I \mid n_R)$
$\mathtt{HDR}_3(\mathsf{mac}(k_m, m_2))$

(a) $m_1 = g^y \mid g^x \mid \mathtt{CKY}_R \mid \mathtt{CKY}_I \mid [\mathtt{SA}_1, \ldots, \mathtt{SA}_n] \mid \mathtt{ID}_R,$
$m_2 = g^x \mid g^y \mid \mathtt{CKY}_I \mid \mathtt{CKY}_R \mid [\mathtt{SA}_1, \ldots, \mathtt{SA}_n] \mid \mathtt{ID}_I.$



Init. I
Resp. R
$n_I, F(cfg_I), \mathtt{ID}_I$
$uid = (n_I, n_R)$
$\mathtt{SA}_R = nego(F(cfg_I), cfg_R)$
$mode = (\mathtt{SA}_R, \mathtt{ID}_I, \mathtt{ID}_R)$
$k_m = \mathsf{kdf}(psk, n_I \mid n_R)$
$n_R, \mathtt{SA}_R, \mathtt{ID}_R, \mathsf{mac}(k_m, - \mid F(cfg_I) \mid \mathtt{ID}_R)$
$uid = (n_I, n_R)$
$mode = (\mathtt{SA}_R, \mathtt{ID}_I, \mathtt{ID}_R)$
$check(cfg_I, mode)$
$k_m = \mathsf{kdf}(psk, n_I \mid n_R)$
$\mathsf{mac}(k_m, - \mid F(cfg_I) \mid \mathtt{ID}_I)$
$complete = \mathsf{true}$
$complete = \mathsf{true}$

(b) F is a formatting function from cfg_I to the payload of the first message that encodes the list of proposals.

Figure 4: IKEv1 aggressive DHE-PSK protocol (a) first messages (b) downgrade protection sub-protocol.

configuration respectively. Note that the only missing component of *mode* is u, pk_I which, conditioned on the responder completing are determined by the initiators configuration.

For the responder, message (log, u, pk_I) authenticates the two public keys and all inputs to *nego*. □

5 Internet Key Exchange

The Internet Key Exchange (IKE) protocol is the key exchange component of the IPsec suite of protocols. Two versions of the protocol are commonly deployed: IKEv1 [28] and IKEv2 [33]. Both variants are inspired by the SIGMA protocol [36] recalled in the introduction, and are believed to inherit its authentication and key-indistinguishability guarantees. Next, we study their downgrade protection sub-protocols.

5.1 IKEv1 does not prevent downgrade attacks

We first consider the DHE-PSK modes of IKEv1, whose first three messages are depicted in Figure 4a. The protocol presumes that both parties can select the pre-shared key (psk) to use from the negotiated security association $\mathtt{SA}_R$ and identifiers ID_I and ID_R; it then confirms that the two parties agree, using a MAC based on psk. The two parties also exchange Diffie-Hellman shares and use them to derive session keys and protect application data but, in 'aggressive' modes, their authentication and downgrade-protection relies solely on the pre-shared key.

The corresponding downgrade protection sub-protocol is depicted in Figure 4b. The initiator begins by extracting a list of supported security associations $[\mathtt{SA}_1, \ldots, \mathtt{SA}_n]$ from its configuration, presumably ordered by preference, formats them (using the function F), and sends them along with a nonce (n_I) to the responder. Each security association specifies a Diffie-Hellman group (for the key exchange); an encryption scheme and a hash algorithm (for protecting messages); and a peer authentication method. The responder chooses one of these associations ($\mathtt{SA}_R$), based on its own configuration, and responds with its own nonce. The initiator checks that this choice is compatible with its proposals, which completes the negotiation. To authenticate one another, to provide key confirmation, and to prevent downgrade attacks, the initiator and responder exchange MACs, optionally signed when using certificates for authentication. For simplicity, Figures 4a and 4b depict the use of just a pre-shared key for authentication. The MACs are computed with a key derived from the pre-shared key and the nonces, over some important parts of the protocol transcript: the key shares, the 8 byte ISAKMP cookies taken from the headers, the client's offered security associations and the sender's identity.

Surprisingly, the MAC does not cover the negotiated security association ($\mathtt{SA}_R$), and this omission leads to a downgrade attack. A man-in-the-middle can simply modify the second message to replace the server's chosen $\mathtt{SA}_R$ with a different $\mathtt{SA}'_R$ compatible with the initiator's proposals. If this new $\mathtt{SA}'_R$ uses an encryption algorithm that the attacker can break (e.g. DES or NULL), then the attacker can break the confidentiality of the first messages sent by the initiator. (Similarly, the first MAC includes ID_R but not ID_I, so an attacker can modify ID_I in the first message, and yet the initiator will complete the sub-protocol without detecting the modification; this is less problematic in the full protocol because IKEv1 continues with a confirmation message from the responder.)

We instantiate our main definitions to IKEv1 to better understand this downgrade-protection failure and propose fixes. Clearly, the protocol offers no authentication guarantees unless the PSKs used by both parties are honest, so we always make that assumption in the following, which enables us to omit the choice of PSKs from the negotiation predicates. In IKEv1, the mac and kdf functions are negotiated as part of $\mathtt{SA}_R$. They are effectively HMAC-MD5 or HMAC-SHA1. For simplicity, we also restrict our attention to clients and servers configured to use only HMAC-SHA1. (See §4 for an explicit handling of cryptographic agility.) We use the following notations for the sub-protocol:

- the goal is to agree on a mode $(\mathtt{SA}_R, ID_I, ID_R)$;
- $cfg_I = (ID_I, [\mathtt{SA}_1, \ldots, \mathtt{SA}_n])$.
- cfg_R includes ID_R and is otherwise unspecified; it would typically also include a list of SAs.
- F is a formatting function from cfg_I to the payload of the first message that encodes the list of proposals above.
- *nego* is a partial function, used by the responder to map $F(cfg_I)$ and cfg_R to some $\mathtt{SA}_R$.
- *check* is used by the initiator to confirm that the mode is acceptable, checking for instance that $\mathtt{SA}_R$ matches one of the initiator's proposals $[\mathtt{SA}_1, \ldots, \mathtt{SA}_n]$.
- $Nego(cfg_I, cfg_R)$, our specification for negotiation, is defined as $(nego(F(cfg_I), cfg_R), cfg_I.ID_I, cfg_R.ID_R)$ when *check* succeeds, and is otherwise undefined.

Our statements and proofs only rely on the properties of F, *nego*, and *check* as stated above, we hence omit a full description.

We first prove partnering security (Definition 3) relying on the security of both kdf (modeled as a PRF keyed with psk) and mac (modeled as a MAC, relying e.g. on existential unforgeability under chosen-message attacks).

To simplify our presentation, we assume in our formal theorem statement a universe of configurations $\mathcal{C}_{fix}$ with fixed kdf and mac algorithms. As discussed above in practice this is HMAC-SHA Consequently, we can rely on standard pseudo-randomness and chosen message attack unforgeability assumptions for kdf and mac rather than their agile security variants which simplifies our theorem and proof. We hope this is useful for a reader accustomed to traditional non-agile provable security developments. This is still meaningful, as configurations may vary for instance on the choice of groups in the full protocol, moreover the proof acts as a template for a fully agile theorem analogous to Theorem 5.

We also assume that the protocol rejects runs in which $ID_I = ID_R$. This is referred to in the literature as the *self-communication* scenario, and in such settings there are well known reflection attacks on IKEv1 [42, 22]. Positive results in this setting would require an extension of our sub-protocol and assumptions about the DH groups or the ISAKMP cookies employed in the protocol.

Theorem 7 (Partnering security of *IKEv1* sub-protocol)**.** *Let $\mathcal{C}_{fix}$ be the universe of configurations used by protocol participants and let PS be such that all PSKs referenced by handles in cfg are honest. Given an adversary $\mathcal{A}$ against the partnering security of IKEv1-sub, we construct adversaries $\mathcal{B}$ and $\mathcal{B}'$ running in about the same time as $\mathcal{A}$ such that $\mathbf{Adv}^{\mathsf{partnering}}_{\mathit{IKEv1\text{-}sub},\,PS}(\mathcal{A})$ is at most*

$$\frac{n^2}{2^{|uid|/2}} + n_p \cdot \mathbf{Adv}^{\mathsf{PRF}}(\mathcal{B}) + n \cdot \mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}(\mathcal{B}') \ ,$$

where n is the number of sessions and n_p is the number different psks employed by sessions.

Proof sketch. Let G_0 be the original partnering security experiment. S is the event that at some point during the experiment, a session π for which $PS(\pi.cfg)$ holds completes without being partnered.

Game G_1 aborts whenever two sessions of the same role assign the same *uid*. We now bound $|\Pr_{\mathsf{G}_0}[S] - \Pr_{\mathsf{G}_1}[S]|$. The length of the client and server nonces is $|uid|/2$ bits. The probability that n such random values give rise to a collision is approximately $n^2 2^{-|uid|/2-1}$. As G_0 and G_1 are equivalent up to collisions, we have $|\Pr_{\mathsf{G}_0}[S] - \Pr_{\mathsf{G}_1}[S]| \leq n^2 2^{-|uid|/2-1}$.

Game G_2 behaves like G_1, except that it keeps a map from pairs of nonces to MAC keys. Instead of defining $k_m = \mathsf{kdf}(psk, n_I \mid n_R)$, game G_2 looks up n_I, n_R in this map to retrieve the corresponding key k_m when it has been defined. Otherwise, it samples a fresh random MAC key k_m and stores the entry (n_I, n_R, k_m) in the map. We bound $|\Pr_{\mathsf{G}_1}[S] - \Pr_{\mathsf{G}_2}[S]| \leq \sum_{i=1}^{n_p} \mathbf{Adv}^{\mathsf{PRF}}(\mathcal{B}_i)$: for each of the n_p pre-shared secrets psk_k we define a hybrid, and bound the probability of $\mathcal{A}$ distinguishing between adjacent hybrids by $\mathbf{Adv}^{\mathsf{PRF}}(\mathcal{B}_i)$. Let $\mathcal{B}$ be a reduction built from the $\mathcal{B}_i$ by picking $i \in [n_p]$ at random and running $\mathcal{B}_i$. The advantage $\mathbf{Adv}^{\mathsf{PRF}}(\mathcal{B})$ is the expected advantage $E[i \leftarrow [n_s]; \mathbf{Adv}^{\mathsf{PRF}}(\mathcal{B}_i)] = \sum_{i=1}^{n_p} \mathbf{Adv}^{\mathsf{PRF}}(\mathcal{B}_i)/n_p$.

We now argue that the probability of event S happening in Game G_2 is upper bounded by the probability of forging a MAC. In Game G_2, the keys k_m are random and when event S holds for a session with $uid = (n_I, n_R)$, the key (n_I, n_R, k_m) was not accessed by any session to generate MACs. Moreover, because $ID_I \neq ID_R$, MACs cannot be reflected. Consequently, any successful adversary in G_2 can be turned into a reduction $\mathcal{B}'$ that guesses the honest PSK handle used by the session for which S holds and returns the MAC received by that session as a forgery. Therefore $\Pr_{\mathsf{G}_2}[S] \leq n \cdot \mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}(\mathcal{B}')$. □

While partnering security can be shown to hold for a very general partnering security predicate, because of the $\mathtt{SA}_R$ spoofing attack the protocol offers provable downgrade protection only for very restrictive configurations. For example, relying on the unambiguous formatting of ID_I and ID_R in the MACed payloads, we have downgrade protection when

1. the client (or the server) uses each PSK only for a fixed sa_R, ID_I, ID_R; or

2. the client proposes only one $\mathtt{SA}$ at a time and checks that the server echoes this proposal in $\mathtt{SA}_R$, and moreover $\mathtt{SA}$ determines ID_I.

A Downgrade Patch for IKEv1

Our analysis of the IKEv1 downgrade-protection sub-protocol suggests an obvious fix: include the mode $(\mathtt{SA}_R, ID_I, ID_R)$ in both MACs. We thus have

$$m_1 = g^y \mid g^x \mid \mathtt{CKY}_R \mid \mathtt{CKY}_I \mid [\mathtt{SA}_1, \ldots, \mathtt{SA}_n] \mid \mathtt{SA}_R, \mathtt{ID}_R, \mathtt{ID}_I$$
$$m_2 = g^x \mid g^y \mid \mathtt{CKY}_I \mid \mathtt{CKY}_R \mid [\mathtt{SA}_1, \ldots, \mathtt{SA}_n] \mid \mathtt{SA}_R, \mathtt{ID}_I, \mathtt{ID}_R.$$

We then obtain downgrade protection under the same conditions as for partnering: that PSKs be honest and both kdf and mac be secure.

Theorem 8 (Downgrade security of patched *IKEv1* sub-protocol)**.** *Let $\mathcal{C}_{fix}$ be the universe of configurations used by protocol participants and let DP be such that $DP(cfg, \cdot)$ implies that all pre-shared keys referenced by handles in cfg are honest. Given an adversary $\mathcal{A}$ against the downgrade security of the sub-protocol, we construct adversaries $\mathcal{B}_i$ and $\mathcal{B}'$ running in about the same time as $\mathcal{A}$ such that $\mathbf{Adv}^{\mathsf{downgrade}}_{IKEv1\text{-}sub,\, DP}(\mathcal{A})$ is at most*

$$\frac{n^2}{2^{|uid|/2}} + n_p \cdot \mathbf{Adv}^{\mathsf{PRF}}(\mathcal{B}) + n \cdot \mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}(\mathcal{B}') \ ,$$

where n is the number of sessions and n_p is the number of different pre-shared keys employed by sessions.

Proof sketch. Let G_0 be the original downgrade security experiment and let S be the event that there exists a downgraded session.

Game G_1 aborts whenever two sessions of the same role assign the same *uid*. Let n be the total number of sessions. The length of client and server nonces is $|uid|/2$ bits. The probability that n such random values give rise to a collision is approximately $n^2 2^{-|uid|/2-1}$. As G_0 and G_1 are equivalent up to collisions, we have $|\Pr_{\mathsf{G}_0}[S] - \Pr_{\mathsf{G}_1}[S]| \leq n^2 2^{-|uid|/2-1}$.

Game G_2 behaves like G_1, except that it keeps a map from pairs of nonces to MAC keys. Instead of defining $k_m = \mathsf{kdf}(psk, n_I \mid n_R)$, game G_2 looks up n_I, n_R in this map to retrieve the corresponding key k_m when it has been defined. Otherwise, it samples a fresh random MAC key k_m and stores the entry (n_I, n_R, k_m) in the map. We bound $|\Pr_{\mathsf{G}_1}[S] - \Pr_{\mathsf{G}_2}[S]| \leq \sum_{i=1}^{n_p} \mathbf{Adv}^{\mathsf{PRF}}(\mathcal{B}_i)$: for each of the n_p pre-shared secrets psk_i we define a hybrid, and bound the probability of $\mathcal{A}$ distinguishing between adjacent hybrids by $\mathbf{Adv}^{\mathsf{PRF}}(\mathcal{B}_i)$. Let $\mathcal{B}$ be a reduction built from the $\mathcal{B}_i$ by picking $i \in [n_p]$ at random and running $\mathcal{B}_i$.

Observe that in G_2, every pair of nonces $(n_I, n_R) = uid$ appears at most once per role, and that peer instances with the same *uid* and configurations that contain only honest pre-shared keys share the same randomly sampled MAC key.

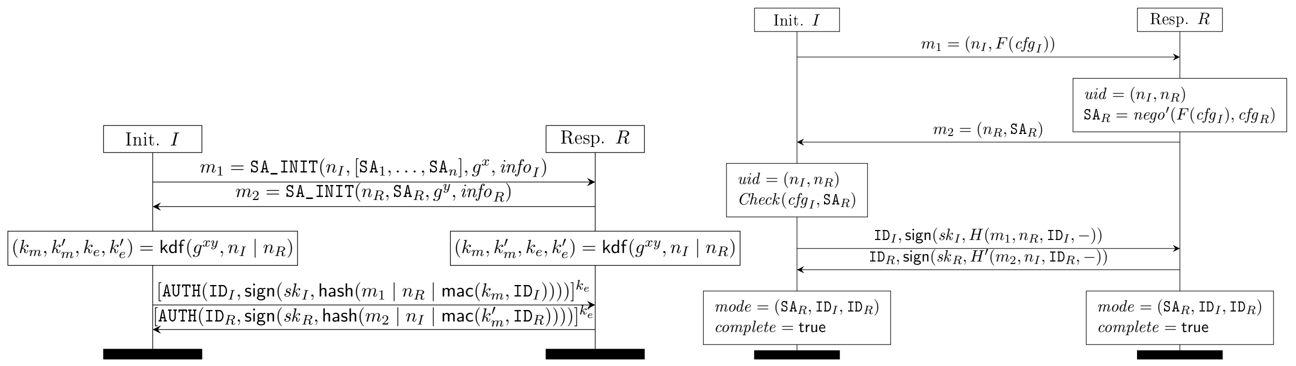
Game G_3 is the same as Game G_2, except that sessions with honest pre-shared keys abort without completing whenever they verify a MAC on m_1 or m_2 that was not computed by another session. We bound the difference between the probability of S in these games using a reduction $\mathcal{B}'$ to the EUF-CMA security of mac:

$$|\Pr_{\mathsf{G}_2}[S] - \Pr_{\mathsf{G}_3}[S]| \leq n \cdot \mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}(\mathcal{B}').$$

Observe that, since the MACs computed by initiators and responders cannot be confounded (because we assumed sessions abort when detecting $ID_I = ID_R$), in Game G_3 we never verify a message that was not authenticated by its unique matching peer.

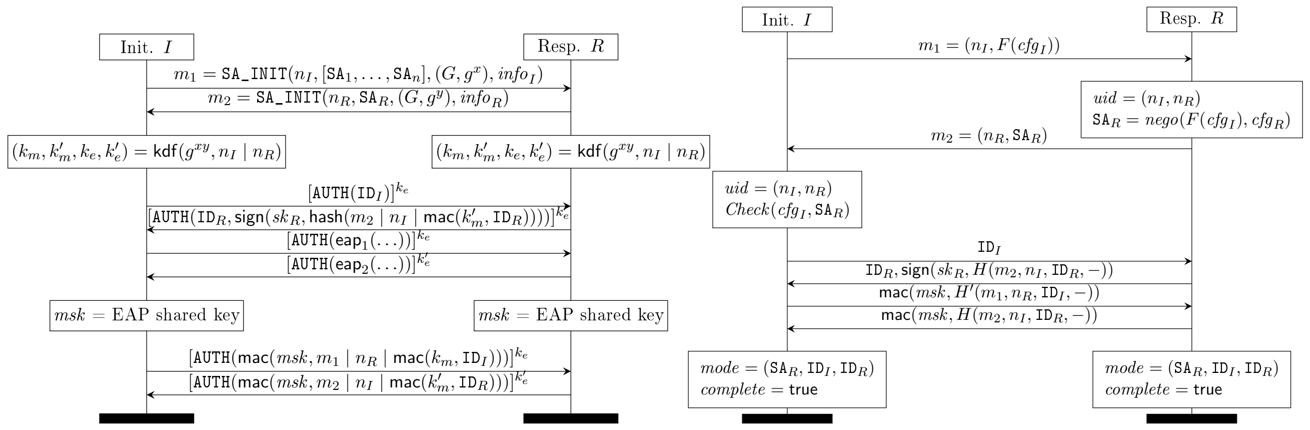
Note that both m_1 and m_2 contain the initiators contribution to *nego* and the responders choice, as well as all other components of *mode*. We thus can exclude the event S from ever occurring in G_3.

□


Init. I
Resp. R
$m_1 = \mathtt{SA\_INIT}(n_I, [\mathtt{SA}_1, \ldots, \mathtt{SA}_n], g^x, info_I)$
$m_2 = \mathtt{SA\_INIT}(n_R, \mathtt{SA}_R, g^y, info_R)$
$(k_m, k'_m, k_e, k'_e) = \mathsf{kdf}(g^{xy}, n_I \mid n_R)$
$(k_m, k'_m, k_e, k'_e) = \mathsf{kdf}(g^{xy}, n_I \mid n_R)$
$[\mathtt{AUTH}(\mathtt{ID}_I, \mathsf{sign}(sk_I, \mathsf{hash}(m_1 \mid n_R \mid \mathsf{mac}(k_m, \mathtt{ID}_I))))]^{k_e}$
$[\mathtt{AUTH}(\mathtt{ID}_R, \mathsf{sign}(sk_R, \mathsf{hash}(m_2 \mid n_I \mid \mathsf{mac}(k'_m, \mathtt{ID}_R))))]^{k'_e}$
Init. I
Resp. R
$m_1 = (n_I, F(cfg_I))$
$uid = (n_I, n_R)$
$\mathtt{SA}_R = nego'(F(cfg_I), cfg_R)$
$m_2 = (n_R, \mathtt{SA}_R)$
$uid = (n_I, n_R)$
$Check(cfg_I, \mathtt{SA}_R)$
$\mathtt{ID}_I, \mathsf{sign}(sk_I, H(m_1, n_R, \mathtt{ID}_I, -))$
$\mathtt{ID}_R, \mathsf{sign}(sk_R, H'(m_2, n_I, \mathtt{ID}_R, -))$
$mode = (\mathtt{SA}_R, \mathtt{ID}_I, \mathtt{ID}_R)$
$complete = \mathsf{true}$
$mode = (\mathtt{SA}_R, \mathtt{ID}_I, \mathtt{ID}_R)$
$complete = \mathsf{true}$

(a) IKEv2 protocol with mutual signatures. (b) IKEv2 sub-protocol with mutual signatures.

Figure 5: IKEv2 protocol and sub-protocol for signature-based authentication


Init. I
Resp. R
$m_1 = \mathtt{SA\_INIT}(n_I, [\mathtt{SA}_1, \ldots, \mathtt{SA}_n], (G, g^x), info_I)$
$m_2 = \mathtt{SA\_INIT}(n_R, \mathtt{SA}_R, (G, g^y), info_R)$
$(k_m, k'_m, k_e, k'_e) = \mathsf{kdf}(g^{xy}, n_I \mid n_R)$
$(k_m, k'_m, k_e, k'_e) = \mathsf{kdf}(g^{xy}, n_I \mid n_R)$
$[\mathtt{AUTH}(\mathtt{ID}_I)]^{k_e}$
$[\mathtt{AUTH}(\mathtt{ID}_R, \mathsf{sign}(sk_R, \mathsf{hash}(m_2 \mid n_I \mid \mathsf{mac}(k'_m, \mathtt{ID}_R))))]^{k'_e}$
$[\mathtt{AUTH}(\mathsf{eap}_1(\ldots))]^{k_e}$
$[\mathtt{AUTH}(\mathsf{eap}_2(\ldots))]^{k'_e}$
msk = EAP shared key
msk = EAP shared key
$[\mathtt{AUTH}(\mathsf{mac}(msk, m_1 \mid n_R \mid \mathsf{mac}(k_m, \mathtt{ID}_I)))]^{k_e}$
$[\mathtt{AUTH}(\mathsf{mac}(msk, m_2 \mid n_I \mid \mathsf{mac}(k'_m, \mathtt{ID}_R)))]^{k'_e}$
Init. I
Resp. R
$m_1 = (n_I, F(cfg_I))$
$uid = (n_I, n_R)$
$\mathtt{SA}_R = nego(F(cfg_I), cfg_R)$
$m_2 = (n_R, \mathtt{SA}_R)$
$uid = (n_I, n_R)$
$Check(cfg_I, \mathtt{SA}_R)$
$\mathtt{ID}_I$
$\mathtt{ID}_R, \mathsf{sign}(sk_R, H(m_2, n_I, \mathtt{ID}_R, -))$
$\mathsf{mac}(msk, H'(m_1, n_R, \mathtt{ID}_I, -))$
$\mathsf{mac}(msk, H(m_2, n_I, \mathtt{ID}_R, -))$
$mode = (\mathtt{SA}_R, \mathtt{ID}_I, \mathtt{ID}_R)$
$complete = \mathsf{true}$
$mode = (\mathtt{SA}_R, \mathtt{ID}_I, \mathtt{ID}_R)$
$complete = \mathsf{true}$

(a) IKEv2 protocol with EAP client authentication. (b) IKEv2 sub-protocol with EAP client authentication.

Figure 6: IKEv2 protocol and sub-protocol for EAP-based authentication

We also considered other modes of IKEv1, based on signatures instead of PSKs (much as in our introductory SIGMA example), and also when the MACs are protected using the keys derived from the Diffie-Hellman exchange. In those cases, the downgrade-protection sub-protocol is almost the same: $\mathtt{SA}_R$ is similarly left unauthenticated and, even if the messages are protected, there is still an attack when the client proposes a weak group, as explained in the introduction.

5.2 IKEv2 does not prevent downgrade attacks

IKEv2 [33] is a revision of the IKEv1 protocol intended to simplify the specification and extend it to cover popular authentication methods such as EAP [1].

5.2.1 IKEv2 with signatures

We first consider the plain, signature-based sub-protocol (see Figure 5b). For brevity, we omit a description of the full protocol (its first messages appear in Figure 8) and we reuse the notations introduced for IKEv1.

We also ignore signature agility issues, since in IKEv2 the hash algorithm for signing is not negotiated; it is chosen by the sender, who almost always picks SHA1.

As in IKEv1, the initiator begins by offering a sequence of security associations (extracted from cfg_I) and the responder chooses one of these. In the full protocol, the initiator and responder also exchange Diffie-Hellman public values and use them to derive session keys, used (in particular) to encrypt and MAC all messages after m_2.

The client and the server then exchange signatures over MACs of their own views of the protocol (presumably to provide some deniability): their full first message, their identity, and the nonce of their peer. In particular, and in contrast with IKEv1, the server's signature covers its chosen $\texttt{SA}_R$ but not the initiator's offered security associations.

The sub-protocol leaves important payloads unauthenticated: the peers do not sign or MAC each other's DH public keys, and not even each other's identities. It also ignores the fact that, in the full protocol, all messages after m_2 are encrypted and MACed using a derived key. Thus, some attacks against the sub-protocol may not occur in the full protocol.

Still, there is a downgrade attack against the full protocol as soon as the client tolerates one weak group. The attack proceeds as follows (see Fig. 7). Suppose an initiator offers two security associations, one using the 1024-bit Diffie-Hellman group 14 and another using the 768-bit group 1. The attacker tampers with the first message to delete the first association, so that the responder thinks that the initiator only supports group 1. The attacker forwards the responder's messages, and now the initiator thinks that the responder only supports group 1. If the attacker has performed enough pre-computation on group 1 so that he can compute the discrete log of a key share, he can then compute the session and MAC keys and impersonate the responder.

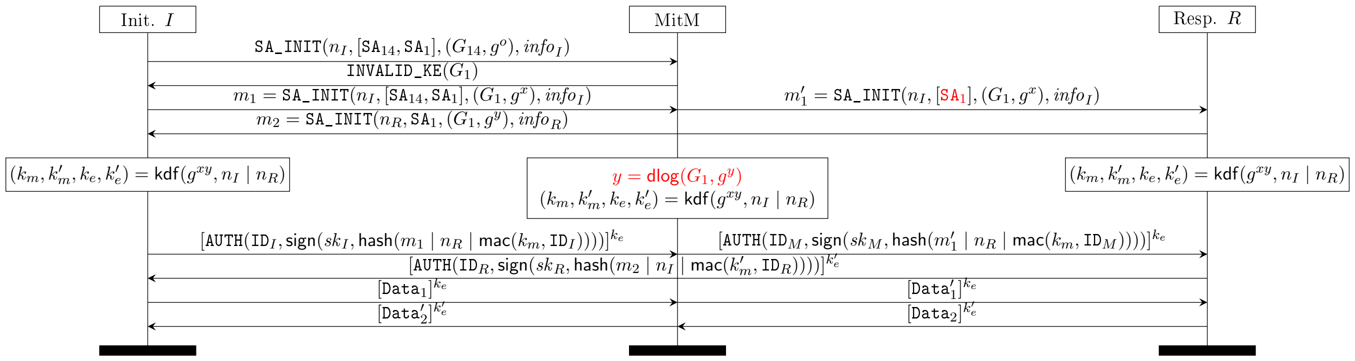

Init. I
MitM
Resp. R
$\texttt{SA\_INIT}(n_I, [\texttt{SA}_{14}, \texttt{SA}_1], (G_{14}, g^o), info_I)$
$\texttt{INVALID\_KE}(G_1)$
$m_1 = \texttt{SA\_INIT}(n_I, [\texttt{SA}_{14}, \texttt{SA}_1], (G_1, g^x), info_I)$
$m'_1 = \texttt{SA\_INIT}(n_I, [\texttt{SA}_1], (G_1, g^x), info_I)$
$m_2 = \texttt{SA\_INIT}(n_R, \texttt{SA}_1, (G_1, g^y), info_R)$
$(k_m, k'_m, k_e, k'_e) = \mathsf{kdf}(g^{xy}, n_I \mid n_R)$
$y = \mathsf{dlog}(G_1, g^y)$
$(k_m, k'_m, k_e, k'_e) = \mathsf{kdf}(g^{xy}, n_I \mid n_R)$
$(k_m, k'_m, k_e, k'_e) = \mathsf{kdf}(g^{xy}, n_I \mid n_R)$
$[\texttt{AUTH}(\texttt{ID}_I, \mathsf{sign}(sk_I, \mathsf{hash}(m_1 \mid n_R \mid \mathsf{mac}(k_m, \texttt{ID}_I))))]^{k_e}$
$[\texttt{AUTH}(\texttt{ID}_M, \mathsf{sign}(sk_M, \mathsf{hash}(m'_1 \mid n_R \mid \mathsf{mac}(k_m, \texttt{ID}_M))))]^{k_e}$
$[\texttt{AUTH}(\texttt{ID}_R, \mathsf{sign}(sk_R, \mathsf{hash}(m_2 \mid n_I \mid \mathsf{mac}(k'_m, \texttt{ID}_R))))]^{k'_e}$
$[\texttt{Data}_1]^{k_e}$
$[\texttt{Data}'_1]^{k_e}$
$[\texttt{Data}'_2]^{k'_e}$
$[\texttt{Data}_2]^{k'_e}$

Figure 7: Man-in-the-middle downgrade on IKEv2 with mutual signatures and weak Diffie-Hellman groups

In practice, executing this attack requires the MitM to send an extra `INVALID_KE` message to the client. This does not present any difficulty since this message is unauthenticated.

The attack described above is reminiscent of Logjam [2] and is arguably feasible with modern computing power, or will be in the coming years. There are other downgrade attacks with a similar impact on IKEv2: the man-in-the-middle could downgrade the security association to use weak encryption or authentication algorithms.

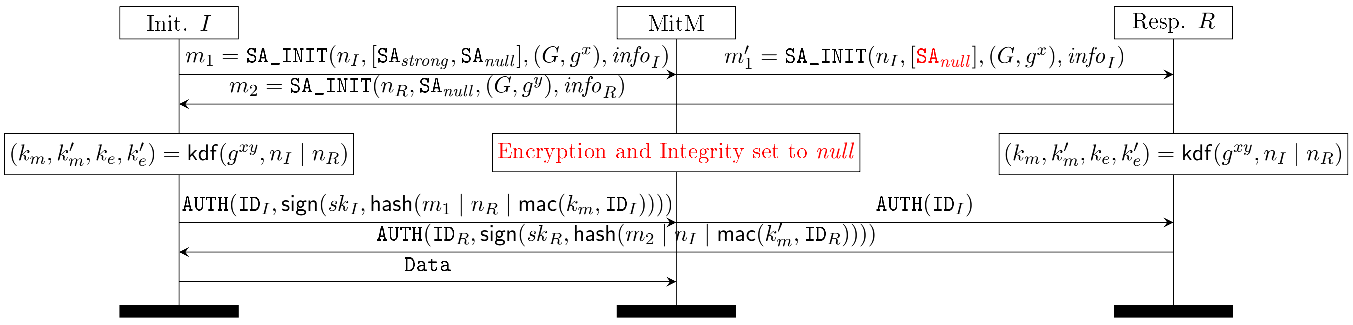

Init. I
MitM
Resp. R
$m_1 = \mathtt{SA\_INIT}(n_I, [\mathtt{SA}_{strong}, \mathtt{SA}_{null}], (G, g^x), info_I)$
$m_1' = \mathtt{SA\_INIT}(n_I, [\mathtt{SA}_{null}], (G, g^x), info_I)$
$m_2 = \mathtt{SA\_INIT}(n_R, \mathtt{SA}_{null}, (G, g^y), info_R)$
$(k_m, k_m', k_e, k_e') = \mathsf{kdf}(g^{xy}, n_I \mid n_R)$
Encryption and Integrity set to *null*
$(k_m, k_m', k_e, k_e') = \mathsf{kdf}(g^{xy}, n_I \mid n_R)$
$\mathtt{AUTH}(\mathtt{ID}_I, \mathsf{sign}(sk_I, \mathsf{hash}(m_1 \mid n_R \mid \mathsf{mac}(k_m, \mathtt{ID}_I))))$
$\mathtt{AUTH}(\mathtt{ID}_I)$
$\mathtt{AUTH}(\mathtt{ID}_R, \mathsf{sign}(sk_R, \mathsf{hash}(m_2 \mid n_I \mid \mathsf{mac}(k_m', \mathtt{ID}_R))))$
Data

Figure 8: Man-in-the-middle cross-protocol downgrade on IKEv2 mixing signatures and EAP authentication

5.2.2 IKEv2 with EAP client authentication

We now consider the downgrade protection sub-protocol in case the initiator is authenticated using some EAP method, whereas the responder still uses a certificate and a signature (see Figure 6b).

In this variant, in the third message, the initiator sends its identity without any signature. Instead, after verifying the server's signature, it engages in an application-level 'embedded' authentication protocol that generates a shared key. Its use of EAP is asymmetric, in that EAP authenticates the initiator (ID_I) but does not re-authenticate the responder. The resulting shared key is used to MAC the initiator's view of the negotiation: the full first message, including the client's offered security associations, the responder's nonce, and a MAC over the initiator's identity with the session key.

Enabling EAP actually weakens downgrade protection: the responder (still) does not sign the initiator's proposals, and also does not sign the chosen client AUTH method (signature or EAP), and this opens the possibility of cross-authentication attacks between different AUTH methods.

For example, consider the attack in Fig. 8. Suppose the initiator disables EAP, but the responder supports it. The attacker can then replace the initiator's signature message with an EAP authentication message, forward the responder's signature, and thereby downgrade the SA used by the initiator, to use a weak encryption algorithm, for instance. In comparison with the first attack on IKEv2 discussed above, this attack does not require breaking the Diffie-Hellman exchange to gain control of the key used to MAC the signature payloads.

This would be a powerful downgrade, and it would allow offline decryption of the initiator's subsequent messages, but it is still difficult to implement in practice because the authentication messages are themselves encrypted-and-MACed. Hence, the attack requires that the attacker should be able to break the (downgraded) authenticated encryption mechanism in the SA.

For example, it can be mounted if the encryption and integrity algorithms are downgraded to NULL, an allowed (but not recommended) option in IKEv2. In particular, the specification says: "Though the security of negotiated Child SAs does not depend on the strength of the encryption and integrity protection negotiated in the IKE SA, implementations MUST NOT negotiate NONE as the IKE integrity protection algorithm or ENCR_NULL as the IKE encryption algorithm." [33, Section 5]. Our attack shows that this assumption is wrong: the downgrade security of IKEv2 crucially depends on the strength of the encryption and integrity algorithms, especially when both signatures and EAP are enabled.

We also note that in case the initiator also supports EAP, any subsequent initiator authentication makes no difference since the initiator is now talking to the attacker and does not seek to re-authenticate the responder.

While these attacks can be mitigated by disabling weak algorithms, or by relying on subsequent key exchanges in Child SAs, a simple protocol-level fix would be for the responder to include the client's first message and authentication mode in its signature (at the cost of losing deniability). We could then obtain downgrade protection simply by relying on the strength of the responder's signature, irrespective of weak groups and broken encryption algorithms.

5.3 Version downgrade attacks from IKEv2 to IKEv1

IKE does not include a version negotiation protocol. Initiators first try to connect with IKEv2 and if that fails they fall back to IKEv1. This allows a simple downgrade attack between this versions, since IKEv1 has no way of authenticating the highest supported version. The IKEv2 specification acknowledges this version downgrade possibility to IKEv1, but sets up a flag to prevent future downgrade attacks from IKEv($n > 2$) to IKEv2: "Note that IKEv1 does not follow these rules, because there is no way in v1 of noting that you are capable of speaking a higher version number. So an active attacker can trick two v2-capable nodes into speaking v1. When a v2-capable node negotiates down to v1, it should note that fact in its logs" [33, Section 2.5].

6 Z Real-Time Protocol

ZRTP [55] is a specialized protocol used to establish key material for encrypted voice-over-IP (VoIP) communications. Unlike TLS, ZRTP does not rely on public-key infrastructure or certificates for authentication. Instead, participants authenticate each other by comparing a "short authentication string" derived from the session key, also known as a SAS, via some trusted channel. For our purposes in this analysis, we assume in our model that the SAS comparison is conducted via an ideal, trusted channel that is not susceptible to tampering.

(Formally we model this using a $\mathsf{CompareSAS}(\pi, \pi')$ oracle that makes session π compare its own sas to that of π', if assigned. Session π must be in a waiting state before this oracle call and continues if the two match, otherwise aborts. We also provide a $\mathsf{LeakSAS}(\pi)$ oracle that marks π's sas as dishonest and leaks it to the adversarym, and a $\mathsf{FakeSAS}(\pi, sas')$ oracle that let the adversary inject its own sas' to be compared with π's sas.)

Because the SAS is short, the protocol offers a more limited form of protection. If the SAS length is ℓ bits, then the probability of an attacker subverting the authentication is at least $2^{-\ell}$ with each execution of the handshake. In most implementations ℓ is typically a small value, e.g. 16. The use of a short authentication string presents challenges for both key exchange and downgrade security. For example, if the SAS employed a full-length collision-resistant hash, it would suffice for the parties to exchange a hash of the full protocol transcript. However, even when constructed using a (truncated) collision-resistant hash function, the SAS is too short to provide the necessary protection, and additional measures must be taken.

6.1 ZRTP does not prevent downgrade attacks

The ZRTP protocol is presented in Figure 9a. The downgrade protection sub-protocol is presented in Figure 9b. The ciphersuite negotiation is conducted within the first two ("Hello") messages exchanged by the Initiator and the Responder. The chosen ciphersuite a_i is determined by selecting a ciphersuite in the intersection of the available algorithms presented by each party. Ciphersuites consist of a key exchange algorithm, a cipher and MAC algorithm for subsequent data exchange, and a SAS algorithm determining

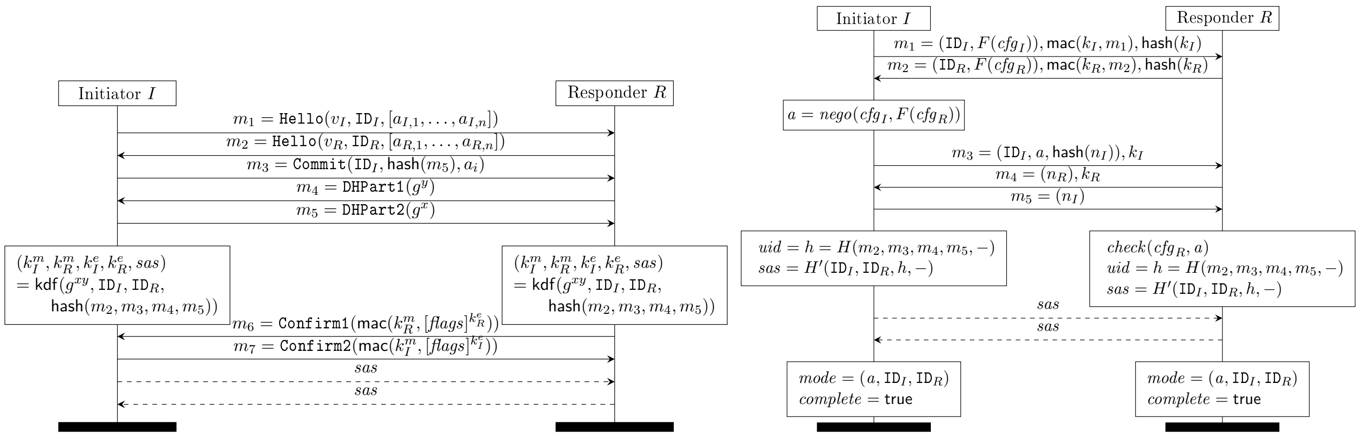

Initiator I
Responder R
$m_1 = \texttt{Hello}(v_I, \mathtt{ID}_I, [a_{I,1}, \dots, a_{I,n}])$
$m_2 = \texttt{Hello}(v_R, \mathtt{ID}_R, [a_{R,1}, \dots, a_{R,n}])$
$m_3 = \texttt{Commit}(\mathtt{ID}_I, \mathsf{hash}(m_5), a_i)$
$m_4 = \texttt{DHPart1}(g^y)$
$m_5 = \texttt{DHPart2}(g^x)$
$(k_I^m, k_R^m, k_I^e, k_R^e, sas) = \mathsf{kdf}(g^{xy}, \mathtt{ID}_I, \mathtt{ID}_R, \mathsf{hash}(m_2, m_3, m_4, m_5))$
$(k_I^m, k_R^m, k_I^e, k_R^e, sas) = \mathsf{kdf}(g^{xy}, \mathtt{ID}_I, \mathtt{ID}_R, \mathsf{hash}(m_2, m_3, m_4, m_5))$
$m_6 = \texttt{Confirm1}(\mathsf{mac}(k_R^m, [\mathit{flags}]^{k_R^e}))$
$m_7 = \texttt{Confirm2}(\mathsf{mac}(k_I^m, [\mathit{flags}]^{k_I^e}))$
sas
sas
(a)
Initiator I
Responder R
$m_1 = (\mathtt{ID}_I, F(cfg_I)), \mathsf{mac}(k_I, m_1), \mathsf{hash}(k_I)$
$m_2 = (\mathtt{ID}_R, F(cfg_R)), \mathsf{mac}(k_R, m_2), \mathsf{hash}(k_R)$
$a = nego(cfg_I, F(cfg_R))$
$m_3 = (\mathtt{ID}_I, a, \mathsf{hash}(n_I)), k_I$
$m_4 = (n_R), k_R$
$m_5 = (n_I)$
$uid = h = H(m_2, m_3, m_4, m_5, -)$
$sas = H'(\mathtt{ID}_I, \mathtt{ID}_R, h, -)$
$check(cfg_R, a)$
$uid = h = H(m_2, m_3, m_4, m_5, -)$
$sas = H'(\mathtt{ID}_I, \mathtt{ID}_R, h, -)$
sas
sas
$mode = (a, \mathtt{ID}_I, \mathtt{ID}_R)$
$complete = \mathsf{true}$
$mode = (a, \mathtt{ID}_I, \mathtt{ID}_R)$
$complete = \mathsf{true}$
(b)

Figure 9: ZRTP (a) protocol (b) downgrade protection sub-protocol

the length and format of the SAS string. Additionally, the protocol negotiates *options* such as a "trusted" PBX flag and an optional signature on the SAS.

Following the initial negotiation messages, the parties determine who will play the role of the Initiator, engage in a key exchange, and derive session keys. Transcript correctness is enforced by incorporating a hash of most of the transcript into the key derivation function, which produces both session keys and a SAS. A final mechanism tries to authenticate each of the handshake messages by computing a MAC over each message, using a key that is revealed in the subsequent message. To bind these messages together, ZRTP uses a hash chain.[1]

Downgrading protocol versions ZRTP includes a negotiation mechanism for protocol versions and options that is not incorporated into the calculation of the shared secrets and SAS. When the parties support multiple versions of the protocol and protocol options, a MitM can substitute the protocol versions v_I, v_R to downgrade both parties to a previous version of the protocol, as illustrated in Figure 10. Moreover, since the first (Initiator Hello) message is not authenticated, the attacker can also change the options flags o_I. This second procedure requires the attacker to defeat the hash chain security mechanism. Unfortunately this may be done by capturing and delaying subsequent messages until the authentication key for earlier messages has been revealed, allowing the attacker to change messages arbitrarily. The fix for this vulnerability is straightforward: all negotiation messages should be included in the calculation of the session key and SAS.

Downgrade from DH to PSK ZRTP supports both Diffie-Hellman key exchange and a pre-shared key mode. The latter is analogous to the session resumption handshake in TLS, in that it provides an inexpensive (symmetric-key only) handshake, which operates under the assumption that the parties have previously completed a full Diffie-Hellman handshake to establish a pre-shared key. The corresponding negotiation sub-protocol is shown in Fig. 11. The limitation of this pre-shared mode is that it does not

[1]Specifically, each participant computes an initial nonce H_0 and hashes it to obtain the sequence $H_3 = \mathsf{hash}(H_2 = \mathsf{hash}(H_1 = \mathsf{hash}(H_0)))$. At each message in the handshake, the party reveals H_i and uses H_{i-1} as a MAC key to authenticate the current message. Verification is only possible when the next message is received. The initial value H_0 is revealed only within the encrypted confirmation message at the conclusion of the protocol.

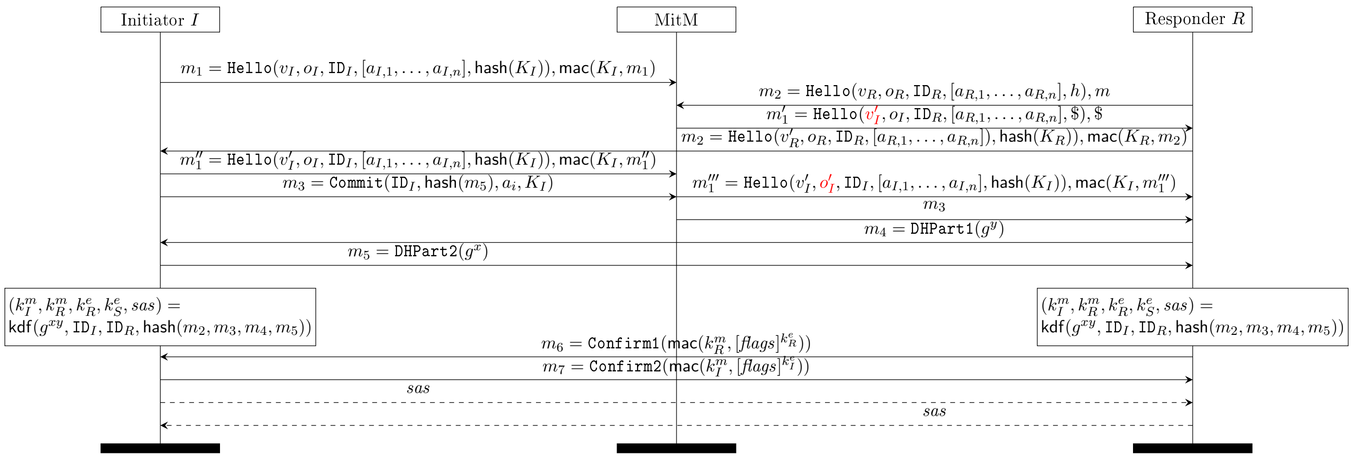

Initiator I
MitM
Responder R
$m_1 = \texttt{Hello}(v_I, o_I, \texttt{ID}_I, [a_{I,1}, \ldots, a_{I,n}], \mathsf{hash}(K_I)), \mathsf{mac}(K_I, m_1)$
$m_2 = \texttt{Hello}(v_R, o_R, \texttt{ID}_R, [a_{R,1}, \ldots, a_{R,n}], h), m$
$m_1' = \texttt{Hello}(v_I', o_I, \texttt{ID}_R, [a_{R,1}, \ldots, a_{R,n}], \$), \$$
$m_2 = \texttt{Hello}(v_R', o_R, \texttt{ID}_R, [a_{R,1}, \ldots, a_{R,n}]), \mathsf{hash}(K_R)), \mathsf{mac}(K_R, m_2)$
$m_1'' = \texttt{Hello}(v_I', o_I, \texttt{ID}_I, [a_{I,1}, \ldots, a_{I,n}], \mathsf{hash}(K_I)), \mathsf{mac}(K_I, m_1'')$
$m_3 = \texttt{Commit}(\texttt{ID}_I, \mathsf{hash}(m_5), a_i, K_I)$
$m_1''' = \texttt{Hello}(v_I', o_I', \texttt{ID}_I, [a_{I,1}, \ldots, a_{I,n}], \mathsf{hash}(K_I)), \mathsf{mac}(K_I, m_1''')$
m_3
$m_4 = \texttt{DHPart1}(g^y)$
$m_5 = \texttt{DHPart2}(g^x)$
$(k_I^m, k_R^m, k_R^e, k_S^e, sas) = \mathsf{kdf}(g^{xy}, \texttt{ID}_I, \texttt{ID}_R, \mathsf{hash}(m_2, m_3, m_4, m_5))$
$(k_I^m, k_R^m, k_R^e, k_S^e, sas) = \mathsf{kdf}(g^{xy}, \texttt{ID}_I, \texttt{ID}_R, \mathsf{hash}(m_2, m_3, m_4, m_5))$
$m_6 = \texttt{Confirm1}(\mathsf{mac}(k_R^m, [\mathit{flags}]^{k_R^e}))$
$m_7 = \texttt{Confirm2}(\mathsf{mac}(k_I^m, [\mathit{flags}]^{k_I^e}))$
sas
sas

Figure 10: Man-in-the-Middle attack on ZRTP version and option negotiation. We assume that both peers prefer version $v_I = v_R$, but will support an older version $v_I' = v_R'$. The attacker additionally modifies the options flags o_I transmitted in m_1.

force the parties to commit to their protocol inputs before revealing them, which admits an offline attack in which a MitM may identify protocol inputs that result in a chosen SAS. A full attack is shown in Figure 12, beginning with the establishment of a shared key (via Diffie-Hellman) and restarting with the PSK mode. In practice, most ZRTP implementations do not implement pre-shared mode, and those that do only allow SAS authentication after DH exchanges. Nevertheless, we hope this protocol-level attack serves as a cautionary tale for future ZRTP implementations and extensions.

7 Transport Layer Security

The Transport Layer Security protocol (TLS) is used to provide secure channels for a variety of Internet applications. It offers a number of key exchange mechanisms, authentication methods, and encryption schemes, so that users can pick and choose mechanisms best suited to their needs.

A negative consequence of this agility is the potential for downgrades. TLS clients and servers commonly support multiple protocol versions and hundreds of ciphersuites, even though some of them are known to be obsolete or even broken. For example, SSL 2 is still supported by 10% of web servers even though it has long been known to be vulnerable to multiple attacks including, notably, a ciphersuite downgrade attack [52] and a dangerous backward compatibility attack [6]. Equally, about 25% of web servers were found to still support export-grade ciphersuites that were deprecated in 2000, enabling powerful downgrade and server impersonation attacks like FREAK [12] and Logjam [2].

Since SSL 3, all versions of TLS incorporate various downgrade protection mechanisms. We will analyze the downgrade protection provided by TLS 1.2 and the proposed improvements in TLS 1.3. In both cases we focus on ephemeral Diffie-Hellman key exchange (DHE/ECDHE).

The message names in the protocol descriptions are inspired by their names in the specifications [5, 47], e.g. the server certificate messages is abbreviated as SC.

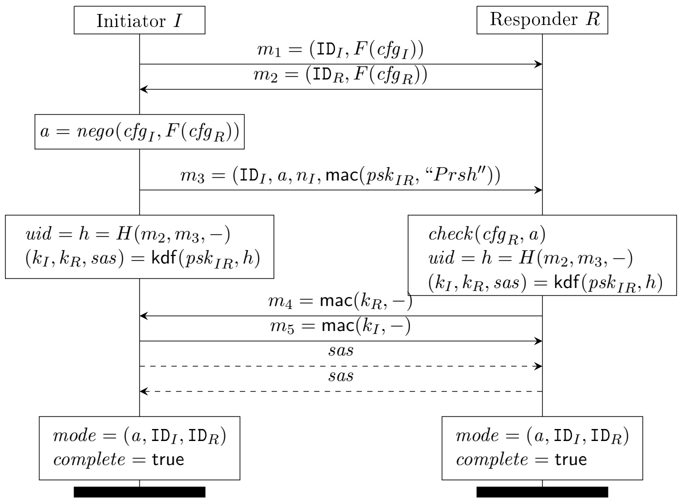

Initiator I
Responder R
$m_1 = (\mathtt{ID}_I, F(cfg_I))$
$m_2 = (\mathtt{ID}_R, F(cfg_R))$
$a = nego(cfg_I, F(cfg_R))$
$m_3 = (\mathtt{ID}_I, a, n_I, \mathsf{mac}(psk_{IR}, \text{“}Prsh''))$
$uid = h = H(m_2, m_3, -)$
$(k_I, k_R, sas) = \mathsf{kdf}(psk_{IR}, h)$
$check(cfg_R, a)$
$uid = h = H(m_2, m_3, -)$
$(k_I, k_R, sas) = \mathsf{kdf}(psk_{IR}, h)$
$m_4 = \mathsf{mac}(k_R, -)$
$m_5 = \mathsf{mac}(k_I, -)$
sas
sas
$mode = (a, \mathtt{ID}_I, \mathtt{ID}_R)$
$complete = \mathsf{true}$
$mode = (a, \mathtt{ID}_I, \mathtt{ID}_R)$
$complete = \mathsf{true}$

Figure 11: ZRTP with Pre-Shared Keys: negotiation sub-protocol

7.1 Negotiation in TLS 1.0 - 1.2

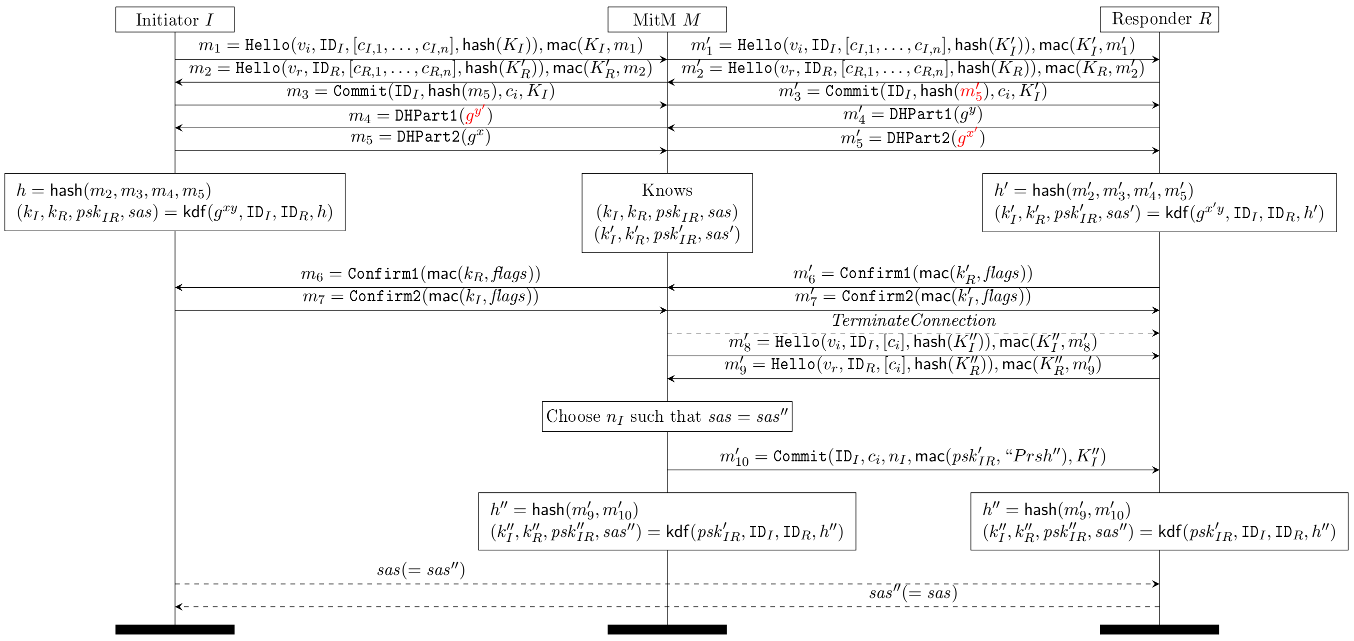
Figure 13a depicts a mutually authenticated TLS connection incorporating a Diffie-Hellman key exchange that uses either a finite-field group (DHE) or an elliptic curve (ECDHE). Most TLS connections authenticate only the server, but the figure also depicts the optional client authentication messages.

The client I first sends a hello message (`CH`) with a nonce (n_I) and a list of agility parameters $[a_1, \ldots, a_n]$ that include ciphersuites, compression methods, and protocol extensions. The server responds with a hello message (`SH`) containing its chosen parameters (a_R). At this point, the client and server know which key exchange they will execute next. In an ephemeral Diffie-Hellman key exchange (DHE/ECDHE), the server sends its public-key certificate ($cert_R$) and uses the private key to sign the nonces, the group (or curve) parameters (p, g) and its own Diffie-Hellman public value (g^y). The server may let the client remain anonymous, or it may require authentication (specifying the class of acceptable certificates $[cert_1, \ldots, cert_m]$), in which case the client sends its own certificate ($cert_I$) and public value (g^x), and uses its private key to sign the full protocol transcript so far (log_1). The client and server then derive a master secret (ms) and session keys (k_1, k_2) from the nonces and shared secret (g^{xy}). To complete the key exchange, both sides compute MACs using the master secret over the protocol transcript, and exchange them in finished messages (`CFIN`,`SFIN`). These MACs provide key confirmation as well as downgrade protection. Once exchanged, the client and server can start exchanging application data encrypted under the new session keys ($[\mathtt{Data}]^k$).

7.2 TLS 1.0 - 1.2 do not prevent downgrades

The downgrade protection sub-protocol for TLS 1.0 - 1.2 is depicted in Fig. 13b. The sub-protocols for TLS 1.0, 1.1, and 1.2 have an almost identical protocol flow and primarily differ in the choice of algorithms. For simplicity, we consider only server-authenticated (EC)DHE connections, where clients are anonymous.

The client offers its entire public configuration ($F(cfg_I)$) to the server, which then computes the negotiated parameters ($mode$) that consist of the protocol version (v), the chosen parameters (a_R), the group (G_R), the server identity (pk_R), and the hash function used in the server signature (hash_1). The protocol version and the ciphersuite in a_R together determine other protocol parameters, such as the key derivation function (kdf), the authenticated encryption scheme, and the MAC and hash functions used in the finished



Initiator I
MitM M
Responder R
$m_1 = \texttt{Hello}(v_i, \texttt{ID}_I, [c_{I,1}, \ldots, c_{I,n}], \mathsf{hash}(K_I)), \mathsf{mac}(K_I, m_1)$
$m'_1 = \texttt{Hello}(v_i, \texttt{ID}_I, [c_{I,1}, \ldots, c_{I,n}], \mathsf{hash}(K'_I)), \mathsf{mac}(K'_I, m'_1)$
$m_2 = \texttt{Hello}(v_r, \texttt{ID}_R, [c_{R,1}, \ldots, c_{R,n}], \mathsf{hash}(K'_R)), \mathsf{mac}(K'_R, m_2)$
$m'_2 = \texttt{Hello}(v_r, \texttt{ID}_R, [c_{R,1}, \ldots, c_{R,n}], \mathsf{hash}(K_R)), \mathsf{mac}(K_R, m'_2)$
$m_3 = \texttt{Commit}(\texttt{ID}_I, \mathsf{hash}(m_5), c_i, K_I)$
$m'_3 = \texttt{Commit}(\texttt{ID}_I, \mathsf{hash}(m'_5), c_i, K'_I)$
$m_4 = \texttt{DHPart1}(g^{y'})$
$m'_4 = \texttt{DHPart1}(g^{y})$
$m_5 = \texttt{DHPart2}(g^{x})$
$m'_5 = \texttt{DHPart2}(g^{x'})$
$h = \mathsf{hash}(m_2, m_3, m_4, m_5)$
$(k_I, k_R, psk_{IR}, sas) = \mathsf{kdf}(g^{xy}, \texttt{ID}_I, \texttt{ID}_R, h)$
Knows
$(k_I, k_R, psk_{IR}, sas)$
$(k'_I, k'_R, psk'_{IR}, sas')$
$h' = \mathsf{hash}(m'_2, m'_3, m'_4, m'_5)$
$(k'_I, k'_R, psk'_{IR}, sas') = \mathsf{kdf}(g^{x'y}, \texttt{ID}_I, \texttt{ID}_R, h')$
$m_6 = \texttt{Confirm1}(\mathsf{mac}(k_R, \mathit{flags}))$
$m_7 = \texttt{Confirm2}(\mathsf{mac}(k_I, \mathit{flags}))$
$m'_6 = \texttt{Confirm1}(\mathsf{mac}(k'_R, \mathit{flags}))$
$m'_7 = \texttt{Confirm2}(\mathsf{mac}(k'_I, \mathit{flags}))$
$\mathit{TerminateConnection}$
$m'_8 = \texttt{Hello}(v_i, \texttt{ID}_I, [c_i], \mathsf{hash}(K''_I)), \mathsf{mac}(K''_I, m'_8)$
$m'_9 = \texttt{Hello}(v_r, \texttt{ID}_R, [c_i], \mathsf{hash}(K''_R)), \mathsf{mac}(K''_R, m'_9)$
Choose n_I such that $sas = sas''$
$m'_{10} = \texttt{Commit}(\texttt{ID}_I, c_i, n_I, \mathsf{mac}(psk'_{IR}, \text{“}Prsh\text{''}), K''_I)$
$h'' = \mathsf{hash}(m'_9, m'_{10})$
$(k''_I, k''_R, psk''_{IR}, sas'') = \mathsf{kdf}(psk'_{IR}, \texttt{ID}_I, \texttt{ID}_R, h'')$
$h'' = \mathsf{hash}(m'_9, m'_{10})$
$(k''_I, k''_R, psk''_{IR}, sas'') = \mathsf{kdf}(psk'_{IR}, \texttt{ID}_I, \texttt{ID}_R, h'')$
$sas(= sas'')$
$sas''(= sas)$

Figure 12: Man-in-the-middle cross-protocol attack on ZRTP with Diffie-Hellman exchange followed immediatly by a pre-shared key exchange: M tampers with the first exchange between I and R and disconnects before they can compare their (differing) SAS values. M then runs a new pre-shared exchange with R and is able to synchronize the SAS values at I and R by finding an appropriate nonce (with 2^{16} work).

messages ($\mathsf{mac}, \mathsf{hash}$). We note that the server may possess several identities and choose one based on the chosen ciphersuite or other protocol extensions offered by the client.

Downgrade protection primarily relies on the MACs in the finished messages, which in turn rely on the strength of the group G_R and the negotiated algorithms kdf, hash, and mac. If a client and server support a weak group, for example, then an attacker can downgrade the group and then break the master secret to forge the MACs, as in Logjam.

A second protection mechanism is the server signature, but we observe that this signature covers only the unique identifier and the group G_R, but none of the other negotiated parameters. For example, the Logjam attacker tricks the server into using an export ciphersuite (DHE-EXPORT) that results in a weak Diffie-Hellman group. The client does not support DHE-EXPORT and still thinks it is using standard DHE, but the attacker can forge the MAC to hide this discrepancy. Importantly, the server signature fails to prevent this attack, because it does not include the ciphersuite. Before this attack was disclosed, many implementations of TLS clients still accepted arbitrary groups.

Furthermore, we note that the negotiated algorithms can be weak in practice. For example, TLS 1.2 supports MD5-based signatures; TLS 1.1 derives keys and transcript hashes based on combinations of MD5 and SHA1. These weak constructions also lead to downgrade and impersonation attacks [13].

Let min_r, max_r be the supported minimum and maximum protocol versions, let $algs_r = [a_1, \ldots, a_m] = F(cfg_I)$ be the ciphersuites and extensions, and let $groups_r$ be the groups supported by role r. In terms of the general definition in Section 2, the downgrade protection sub-protocol uses the following session

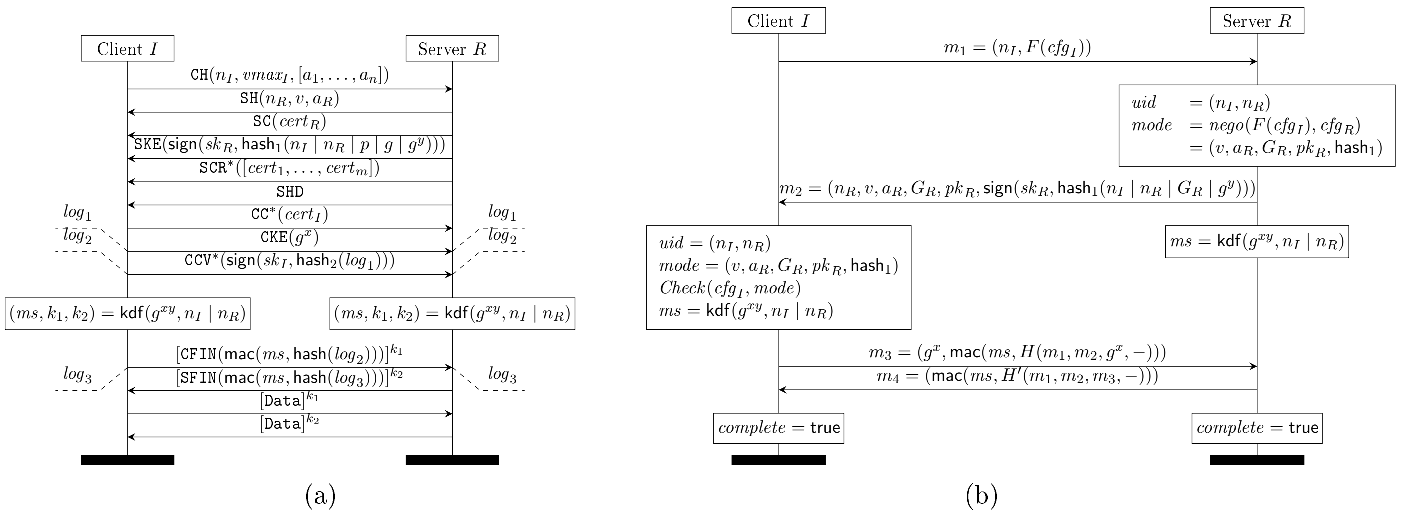

Client I
Server R
$\mathtt{CH}(n_I, vmax_I, [a_1, \ldots, a_n])$
$\mathtt{SH}(n_R, v, a_R)$
$\mathtt{SC}(cert_R)$
$\mathtt{SKE}(\mathsf{sign}(sk_R, \mathsf{hash}_1(n_I \mid n_R \mid p \mid g \mid g^y)))$
$\mathtt{SCR}^*([cert_1, \ldots, cert_m])$
$\mathtt{SHD}$
$\mathtt{CC}^*(cert_I)$
$\mathtt{CKE}(g^x)$
$\mathtt{CCV}^*(\mathsf{sign}(sk_I, \mathsf{hash}_2(log_1)))$
log_1
log_2
$(ms, k_1, k_2) = \mathsf{kdf}(g^{xy}, n_I \mid n_R)$
$[\mathtt{CFIN}(\mathsf{mac}(ms, \mathsf{hash}(log_2)))]^{k_1}$
$[\mathtt{SFIN}(\mathsf{mac}(ms, \mathsf{hash}(log_3)))]^{k_2}$
log_3
$[\mathtt{Data}]^{k_1}$
$[\mathtt{Data}]^{k_2}$
(a)
$m_1 = (n_I, F(cfg_I))$
$uid = (n_I, n_R)$
$mode = nego(F(cfg_I), cfg_R) = (v, a_R, G_R, pk_R, \mathsf{hash}_1)$
$m_2 = (n_R, v, a_R, G_R, pk_R, \mathsf{sign}(sk_R, \mathsf{hash}_1(n_I \mid n_R \mid G_R \mid g^y)))$
$ms = \mathsf{kdf}(g^{xy}, n_I \mid n_R)$
$mode = (v, a_R, G_R, pk_R, \mathsf{hash}_1)$
$Check(cfg_I, mode)$
$m_3 = (g^x, \mathsf{mac}(ms, H(m_1, m_2, g^x, -)))$
$m_4 = (\mathsf{mac}(ms, H'(m_1, m_2, m_3, -)))$
$complete = \mathsf{true}$
(b)

Figure 13: TLS 1.0 - 1.2 with (EC)DHE key exchange (a), where messages labeled with * occur only when client authentication is enabled, and (b) its downgrade protection sub-protocol

variables:

$$\begin{aligned}
cfg &\triangleq \begin{cases} (I, min_I, max_I, algs_I, groups_I, PKs_R) & \text{for } I \\ (R, min_R, max_R, algs_R, groups_R, PK_R) & \text{for } R \end{cases} \\
uid &\triangleq (n_I, n_R) \\
mode &\triangleq (v, a_R, G_R, pk_R, \mathsf{hash}_1)
\end{aligned}$$

The negotiation function *nego* is executed by the server and is based on the server's configuration cfg_R and the server's partial view $F(cfg_I)$ of the client configuration. The client does not get to inspect cfg_R, but it does check that the resulting mode is consistent with its configuration.

The protocol only offers downgrade protection if the peer is authenticated with an honest key and strong signature and hash algorithms. So we will consider downgrade security from the viewpoint of a client, while assuming that all keys in PKs_R are honest and hash_1 is collision-resistant. We get partnering security from the freshness of the *uid* and the strength of the server signature (which includes the *uid*).

However, downgrade protection for the client cannot rely on just the signature, and hence requires one of the following conditions:

- the server uses its pk_R only with modes that use strong groups, key derivation algorithm kdf, hash and mac algorithms and the client is aware of the servers choice and aborts whenever it sees an unexpected algorithm combination;

- the client only accepts modes with strong groups (in particular *not* the groups 'negotiated' by the Logjam and the ECDHE-DHE cross-protocol attacks [41]) and algorithms.

An extreme example of the first condition would be to require that the server uses a different public key for each mode; the proofs in [24] rely on this somewhat unrealistic assumption to avoid ECDHE-DHE cross-protocol attacks and the need for agile security assumptions. More pragmatically, if a client and server only support TLS 1.2 (and hence only strong hash constructions), only support strong groups and curves for (EC)DHE and all other ciphersuites that use Diffie-Hellman, then TLS clients can be protected

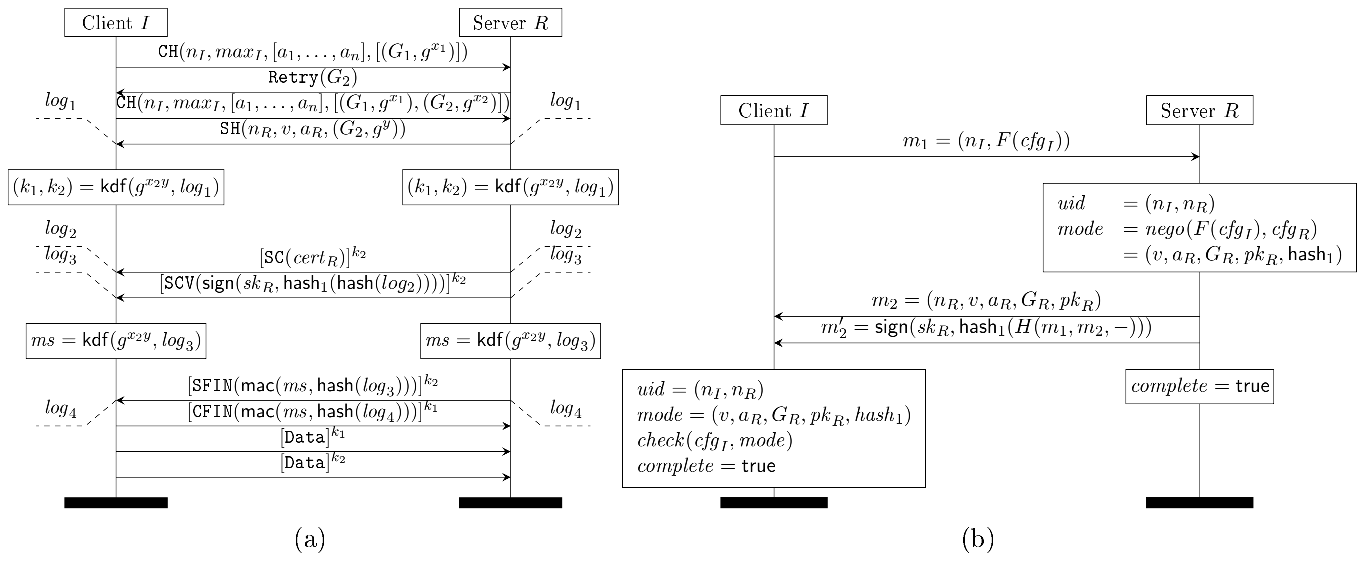

Client I
Server R
$\mathtt{CH}(n_I, max_I, [a_1,\ldots,a_n], [(G_1, g^{x_1})])$
$\mathtt{Retry}(G_2)$
log_1
$\mathtt{CH}(n_I, max_I, [a_1,\ldots,a_n], [(G_1, g^{x_1}), (G_2, g^{x_2})])$
log_1
$\mathtt{SH}(n_R, v, a_R, (G_2, g^y))$
$(k_1,k_2) = \mathsf{kdf}(g^{x_2 y}, log_1)$
$(k_1,k_2) = \mathsf{kdf}(g^{x_2 y}, log_1)$
log_2
log_3
$[\mathtt{SC}(cert_R)]^{k_2}$
log_2
log_3
$[\mathtt{SCV}(\mathsf{sign}(sk_R, \mathsf{hash}_1(\mathsf{hash}(log_2))))]^{k_2}$
$ms = \mathsf{kdf}(g^{x_2 y}, log_3)$
$ms = \mathsf{kdf}(g^{x_2 y}, log_3)$
$[\mathtt{SFIN}(\mathsf{mac}(ms, \mathsf{hash}(log_3)))]^{k_2}$
log_4
$[\mathtt{CFIN}(\mathsf{mac}(ms, \mathsf{hash}(log_4)))]^{k_1}$
log_4
$[\mathtt{Data}]^{k_1}$
$[\mathtt{Data}]^{k_2}$
(a)
Client I
Server R
$m_1 = (n_I, F(cfg_I))$
$uid = (n_I, n_R)$
$mode = nego(F(cfg_I), cfg_R)$
$= (v, a_R, G_R, pk_R, \mathsf{hash}_1)$
$m_2 = (n_R, v, a_R, G_R, pk_R)$
$m_2' = \mathsf{sign}(sk_R, \mathsf{hash}_1(H(m_1, m_2, -)))$
$uid = (n_I, n_R)$
$mode = (v, a_R, G_R, pk_R, hash_1)$
$check(cfg_I, mode)$
$complete = \mathsf{true}$
$complete = \mathsf{true}$
(b)

Figure 14: TLS 1.3 1-RTT mode with server-only authentication (a) and its downgrade protection sub-protocol (b)

from downgrade. Of course, we rely on the server using only honest and strong signing keys (e.g. 2048-bit RSA) with strong signature and hash algorithms (e.g. RSA-SHA256).

We also get some downgrade protection for the server when the client is authenticated, relying only on the client signature and the transcript hash algorithm hash. Pragmatically, TLS 1.2 servers that require client authentication and only accept strong signature and hash algorithms cannot themselves be tricked into completing a connection with a weak *mode*.

As evidenced by the Logjam attack, the TLS protocol does not satisfy downgrade security unless the DP predicate guarantees that the client and server configurations exclusively use strong algorithms, hence guaranteeing that all the negotiated algorithms used in the finished MACs are strong.

7.3 On downgrade protection in Draft 10 of TLS 1.3

Draft 10 of TLS 1.3, the next version of TLS, proposes a protocol that is quite different from TLS 1.2 and earlier versions; a typical run of the 1-round-trip mode is depicted in Fig. 14a. The corresponding downgrade protection sub-protocol is in Fig. 14b.

In contrast to TLS 1.2, the client hello message includes Diffie-Hellman public values for the client's preferred groups. The server may choose one of these groups or ask for a public value in a different group, as long as it is one supported by the client. The server sends its own public value in the server hello message, and all subsequent messages are encrypted and integrity-protected using the Diffie-Hellman shared key.

For downgrade security from the client's viewpoint, a key difference is that server signatures in TLS 1.3 cover the full transcript, and hence they cover the full client and server hello messages. This foils most of the downgrade attacks on TLS 1.2; as long as the client only accepts strong signature and hash algorithms and honest public keys from the server, it cannot be downgraded to a weaker ciphersuite, and moreover, it yields agreement on the chosen ciphersuite.

Although Draft 10 of TLS 1.3 provides strong downgrade protection for the ciphersuite, downgrade attacks remain, in particular, because clients and servers will continue to support lower protocol versions for backward compatibility. Considering that TLS 1.2 does not provide strong downgrade protections, this unfortunately means that all the downgrade attacks on TLS 1.2 will be inherited by TLS 1.3.

There are three downgrade attacks possible on TLS 1.3 as described in Draft 10. One, an attacker downgrades the connection to TLS 1.2 or lower and mounts any of the downgrade attacks mentioned before. This will succeed as long as the attacker can forge the finished MACs. Second, an attacker uses the TLS fallback mechanism to stop TLS 1.3 connections and allows only TLS 1.2 connections to go through. Even if the endpoints implement the fallback protection mechanism [45], the attacker can use one of the downgrade attacks in TLS 1.2 to break the connection. Third, in Draft 10 of the TLS 1.3 protocol, the handshake hashes restart upon receiving a `Retry` message and hence, the attacker can downgrade the Diffie-Hellman group for some classes of negotiation functions.

We can prevent all of these attacks by two countermeasures, both of which have been incorporated into TLS 1.3 Draft 11. See Fig. 15a. First, we continue the handshake hashes over retries. Second, TLS 1.3 servers always include their highest supported version number in the server nonce, even when they choose a lower version such as TLS 1.0.

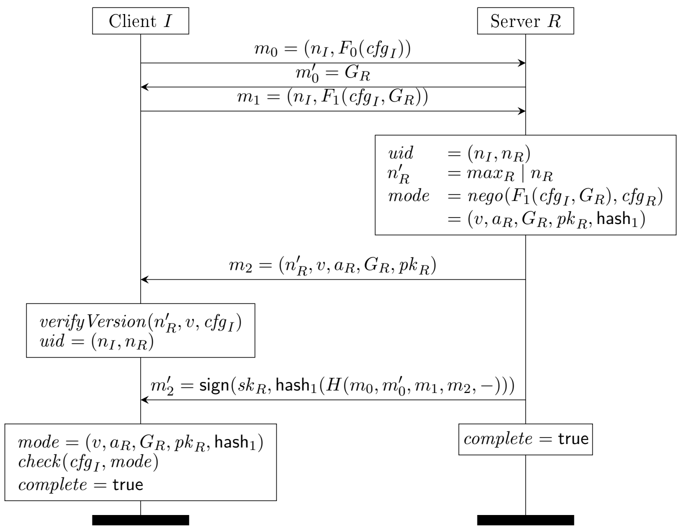
Including the maximum version number into the server nonce of all versions yields *version* downgrade protection for clients. It is a simple patch (For the server, it amounts to changing how nonces are generated. The client needs to implement an equality check.) that can be incorporated into TLS versions without making them incompatible with TLS versions that do not implement the patch. If a server and a client both implement the patch, the client gets version downgrade protection.

We proceed in three steps: We show that when hashes continue over `Retry`, clients that interact with servers that just support TLS 1.3 achieve downgrade security. We then show that embedding the version number into the server's nonce yields *version* downgrade protection from the client's perspective. We then put the two results together and show that servers supporting TLS 1.0 - 1.3 with these countermeasures yield the same client-side downgrade protection as when servers just support TLS 1.3.

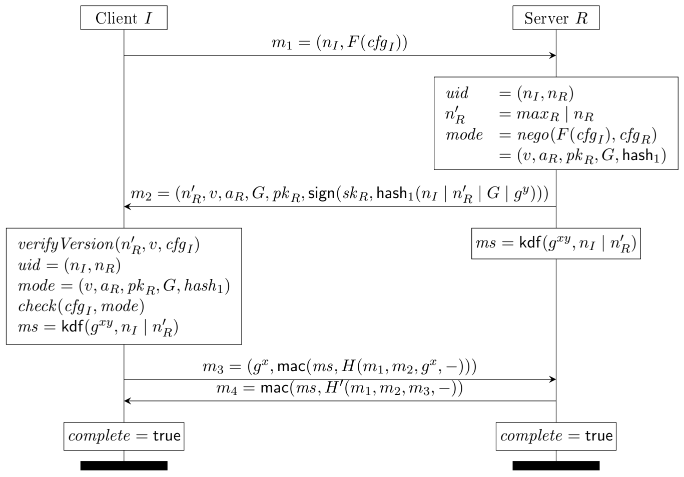
The downgrade protection sub-protocol uses the same session variables as TLS 1.0 - 1.2, but defines Nego using the function nego from Fig. 15a. Let $\mathcal{M}$ be the set of modes supported by TLS and $\mathcal{M}^\star = \{\mathit{Nego}_{\mathit{cfg.role}}(\mathit{cfg}, \mathit{cfg}') | \mathit{PS}(\mathit{cfg})\}$ be the modes negotiated between any pair of configurations for which the first guarantees partnering security. Let $\mathcal{P}_s = \{p \mid s, p = \mathit{mode}.\overline{\mathit{sig}} \wedge \mathit{mode} \in \mathcal{M}\}$ be the signature agility parameters for peer signature scheme s, $\mathcal{H}$ be the set of all hash algorithms supported by TLS, and $\mathcal{H}^\star = \{\mathit{mode.hash} \mid \mathit{mode} \in \mathcal{M}^\star\}$ be the hash algorithms used by partnering secure modes. We now prove partnering security for TLS 1.0 - 1.3 configurations that only enable (EC)DHE, and downgrade security for clients speaking to servers that implement the fix described in Fig. 15a. We then define version downgrade security and show that the fixes in Fig. 15a (TLS 1.3) and Fig. 15b (TLS 1.0 - 1.2) prevent version downgrade.

Theorem 9 (Partnering security of *TLS1.0-1.3*-sub)**.** *We consider a universe of configurations where RSA keys are used for signing only, where only (EC)DHE is enabled, and where the configurations support some subset of versions of TLS 1.0, 1.1, 1.2 and 1.3. Let PS be such that $\mathit{PS}(\mathit{cfg})$ implies that $\mathit{cfg.role} = I$, that all public keys in the range of $\mathit{cfg.PKs}_R$ are honest and that cfg only supports (EC)DHE. Given an adversary $\mathcal{A}$ against the partnering security of TLS, we construct adversaries $\mathcal{B}_{s,p,i}$ and $\mathcal{B}_h$ running in about the same time as $\mathcal{A}$ such that $\mathbf{Adv}^{\mathsf{partnering}}_{\mathit{TLS1.0\text{-}1.3\text{-}sub},\,PS}(\mathcal{A})$ is at most*

$$\sum_{h \in \mathcal{H}^\star} \mathbf{Adv}^{\mathsf{CR}}_{h,\mathcal{H}}(\mathcal{B}_h) + \sum_{(s,p)\in\overline{\mathit{sig}}(\mathcal{M}^\star)} \sum_{i=1}^{n_s} \mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}_{s,\,p,\,\mathcal{P}_s}(\mathcal{B}_{s,p,i}) \,,$$


Client I
Server R
$m_0 = (n_I, F_0(cfg_I))$
$m'_0 = G_R$
$m_1 = (n_I, F_1(cfg_I, G_R))$
$uid = (n_I, n_R)$
$n'_R = max_R \mid n_R$
$mode = nego(F_1(cfg_I, G_R), cfg_R) = (v, a_R, G_R, pk_R, \mathsf{hash}_1)$
$m_2 = (n'_R, v, a_R, G_R, pk_R)$
$verifyVersion(n'_R, v, cfg_I)$
$uid = (n_I, n_R)$
$m'_2 = \mathsf{sign}(sk_R, \mathsf{hash}_1(H(m_0, m'_0, m_1, m_2, -)))$
$mode = (v, a_R, G_R, pk_R, \mathsf{hash}_1)$
$check(cfg_I, mode)$
$complete = \mathsf{true}$
$complete = \mathsf{true}$

(a) A version downgrade fix for TLS 1.3


Client I
Server R
$m_1 = (n_I, F(cfg_I))$
$uid = (n_I, n_R)$
$n'_R = max_R \mid n_R$
$mode = nego(F(cfg_I), cfg_R) = (v, a_R, pk_R, G, \mathsf{hash}_1)$
$m_2 = (n'_R, v, a_R, G, pk_R, \mathsf{sign}(sk_R, \mathsf{hash}_1(n_I \mid n'_R \mid G \mid g^y)))$
$verifyVersion(n'_R, v, cfg_I)$
$uid = (n_I, n_R)$
$mode = (v, a_R, pk_R, G, hash_1)$
$check(cfg_I, mode)$
$ms = \mathsf{kdf}(g^{xy}, n_I \mid n'_R)$
$ms = \mathsf{kdf}(g^{xy}, n_I \mid n'_R)$
$m_3 = (g^x, \mathsf{mac}(ms, H(m_1, m_2, g^x, -)))$
$m_4 = \mathsf{mac}(ms, H'(m_1, m_2, m_3, -))$
$complete = \mathsf{true}$
$complete = \mathsf{true}$

(b) A version downgrade fix for TLS 1.0 - 1.2

where n_s keys are generated for signing scheme s.

The proof is the same as for Theorem 5 for SSH except that we only need to prove the property for clients and rely on the nonces being hashed and signed by the responder.

For downgrade security, we define $Nego$, $\mathcal{M}$, $\mathcal{P}_s$, and $\mathcal{H}$ as before. However, we redefine $\mathcal{M}^\star$, $\mathcal{H}^\star$ to use DP instead of PS, i.e., $\mathcal{M}^\star = \{Nego_{cfg.role}(cfg, cfg') \mid DS(cfg, cfg')\}$ and $\mathcal{H}^\star = \{mode.hash \mid mode \in \mathcal{M}^\star\}$. Let TLS 1.$x$...1.3 be a version range from TLS 1.x, $x \in [0, 3]$, ending in TLS 1.3. Version TLS 1.3 needs to be always present as our version downgrade guarantees concern downgrades from TLS 1.3 to lower versions.

Theorem 10 (Downgrade security of *TLS1.3-sub*)**.** *We consider a universe of configurations where RSA keys are used for signing or encryption, but not for both.*

Let DP be such that $DP(cfg, cfg')$ implies that $cfg.role = I$, that all public keys in the range of $cfg.PKs_R$ are honest and such that: (a) cfg supports some version range TLS 1.x...1.3, only supports (EC)DHE, and implements the countermeasure. (b) cfg' only supports TLS 1.3 and implement the countermeasure. Given an adversary $\mathcal{A}$ against the downgrade security of TLS1.3-sub, we construct adversaries $\mathcal{B}_{s,p,i}$ and $\mathcal{B}_h$ running in about the same time as $\mathcal{A}$ such that $\mathbf{Adv}^{\mathsf{downgrade}}_{TLS1.3\text{-}sub,\, DP}(\mathcal{A})$ is at most

$$\frac{n^2}{2^{|n_R|+1}} + \sum_{h \in \mathcal{H}^\star} \mathbf{Adv}^{\mathsf{CR}}_{h,\mathcal{H}}(\mathcal{B}_h) + \sum_{(s,p)\in \overline{sig}(\mathcal{M}^\star)} \sum_{i=1}^{n_s} \mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}_{s,\,p,\,\mathcal{P}_s}(\mathcal{B}_{s,p,i}) \,,$$

where n is the number of sessions, n_s is the number of keys generated for signing scheme s, and $|n_R|$ is the size of the servers contribution to the unique identifiers. The current proposal is 24 bytes.

The proof again follows the same structure as for Theorem 6 for SSH except that we only need to prove the property for clients.

Proof sketch. Game G_0 is the original downgrade experiment. Game G_1 aborts if there is at most one partnered server session. By the collision probability of server nonces we have that $|\Pr_{\mathsf{G}_0}[S] - \Pr_{\mathsf{G}_1}[S]| \leq$

$n^2(2^{-|n_R|-1})$. In games G_2 and G_3 we abort on agile hash function collisions and signature forgeries for modes in $\mathcal{M}^\star$.

The final reasoning step follows a different pattern than SSH. As the server signs all information that is taken into account by *nego* together with the corresponding nonces, the signature over the protocol messages must have been generated by the server session with the unique identifier corresponding to these nonces. Moreover, if there had been any modification to the messages of the server and the client that influence *nego*, then the client would have aborted in Game G_3. □

We define version downgrade security similarly to downgrade security via a function $\mathit{Version}_r$ that maps two opposite-role configurations (which include the version numbers) to the version number negotiated (if any) in the absence of active adversaries. Formally, if a session π talking to a session π' completes, it must be the case that $\pi.v = \mathit{Version}_r(\mathit{cfg}_r, \mathit{cfg}_{\bar{r}})$. Akin to downgrade security, our definition of version downgrade security is parameterized by a version downgrade protection predicate *VDP* on pairs of configurations. When $\mathit{VDP}(\mathit{cfg}_r, \mathit{cfg}_{\bar{r}})$ holds, we expect that the local session r is protected. For TLS, we will only consider version downgrade protection from the client's perspective.

Definition 14 (Version downgrade security)**.** *The advantage* $\mathbf{Adv}^{\mathsf{version}}_{\Pi,\,\mathit{VDP}}(\mathcal{A})$ *of* $\mathcal{A}$ *against the version downgrade security of* Π *is the probability that, when* $\mathcal{A}$ *terminates after interacting with protocol* Π *through its oracles, there exists a session* π *such that* $\pi.\mathit{complete} = \mathsf{true}$ *and there is a partnered session* π' *such that* $\mathit{VDP}(\pi.\mathit{cfg}, \pi'.\mathit{cfg})$ *but* $\pi.v \neq \mathit{Version}_{\pi.\mathit{role}}(\pi.\mathit{cfg}, \pi'.\mathit{cfg})$.

Note that this notion of version downgrade security assures agreement on the version number.

Theorem 11 (Version downgrade security of *TLS1.0-TLS1.3-sub*)**.** *We consider a universe of configurations where RSA keys are used for signing or encryption, but not for both. Let VDP be such that* $\mathit{VDP}(\mathit{cfg}, \mathit{cfg}')$ *implies that cfg.role = I and that all public keys in the range of* $\mathit{cfg}.\mathit{PKs}_R$ *are honest and such that: (a) cfg only supports (EC)DHE and implements the countermeasure for the version range TLS 1.x...1.3 that it supports. (b) cfg′ activates the countermeasure for the version range TLS 1.x...1.3 that is supports.*

Given an adversary $\mathcal{A}$ *against the version downgrade security of TLS1.0-TLS1.3-sub, we construct adversaries* $\mathcal{B}_{s,p,i}$ *and* $\mathcal{B}_h$ *running in about the same time as* $\mathcal{A}$ *such that* $\mathbf{Adv}^{\mathsf{version}}_{\mathit{TLS1.0\text{-}TLS1.3\text{-}sub},\,\mathit{VDP}}(\mathcal{A})$ *is at most*

$$\frac{n^2}{2^{|n_R|+1}} + \sum_{h \in \mathcal{H}^\star} \mathbf{Adv}^{\mathsf{CR}}_{h,\,\mathcal{H}}(\mathcal{B}_h) + \sum_{(s,p) \in \overline{sig}(\mathcal{M}^\star)} \sum_{i=1}^{n_s} \mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}_{s,\,p,\,\mathcal{P}_s}(\mathcal{B}_{s,p,i}) \;,$$

where n is the number of sessions, n_s *is the number of keys generated for signing scheme s, and* $|n_R|$ *is the size of the servers contribution to the unique identifiers. The current proposal is 24 bytes. The client also gets*

Proof sketch. Let S be the event that there exists a session π that completed and is version downgraded, i.e., there exists a partner session π' with the same *uid* and opposite role and such that $\pi.v \neq \mathit{Version}_{\pi.\mathit{role}}(\pi.\mathit{cfg}, \pi'.\mathit{cfg})$

Let G_0 be the original version downgrade security experiment. Game G_1 aborts whenever two server sessions assign the same n_R. Let n be the total number of sessions. The probability that n such random values give rise to a collision is upper bounded by $n^2 2^{-|n_R|-1}$. As G_0 and G_1 are identical up to collisions, we have $|\Pr_{\mathsf{G}_0}[S] - \Pr_{\mathsf{G}_1}[S]| \leq n^2 2^{-|n_R|/2-1}$. Game G_2 behaves as G_1, except that it aborts whenever two sessions hash different transcripts to the same *log*. Using $\mathcal{A}$, we can construct for each $h \in \mathcal{H}^\star$ an adversary

$\mathcal{B}_h$ such that $|\Pr_{\mathsf{G}_0}[S] - \Pr_{\mathsf{G}_1}[S]| \leq \sum_{h \in \mathcal{H}^\star} \mathbf{Adv}^{\mathsf{CR}}_{h,\mathcal{H}}(\mathcal{B}_h)$. Observe that in G_2, every n_R is signed at most once by the responder.

Game G_3 is the same as Game G_2, except that sessions with honest peer keys and $(s,p) \in \overline{sig}(\mathcal{M})$ abort without completing whenever they verify a signature that was not produced by another session.

We again bound the difference in the success probabilities using the lemma of Bhargavan et al. [16, Lemma 4] and have

$$|\Pr_{\mathsf{G}_2}[S] - \Pr_{\mathsf{G}_3}[S]| \leq \sum_{(s,p)\in\overline{sig}(\mathcal{M}^\star)} \sum_{i=1}^{n_s} \mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}_{s,\,p,\,\mathcal{P}_s}(\mathcal{B}_{s,p,i}).$$

Observe that, since the signatures computed by initiators and responders cannot be confounded, in Game G_3 we never verify a message that was not signed by its unique partner.

For the initiator we have that message *log* was signed by its partner and contains the responder's nonce. As the *VDP* holds, the responder's nonce contains the maximal supported version number and the initiator verifies that this matches the one that the initiator either proposed or received, the initiator yields the preferred protocol version.

□

For a downgrade predicate *DP* and a version predicate *VDP* such that $DP \subseteq VDP$, let DP^{+VDP} be the predicate that holds for pair of configurations in *DP*, with server configurations extended to also support configurations of lower version protocols that by *VDP* should never be negotiated. Putting Theorem 11 and Theorem 10 together, we get that when both client and server implement the countermeasures, then from the perspective of the client, supporting a range TLS 1.x...1.3 is as good as supporting only TLS 1.3.

Corollary 1 (Downgrade security of *TLS1.0-TLS1.3-sub*)**.** *We consider a universe of configurations where only (EC)DHE is enabled and RSA keys are used for signing only. Let $DP \subseteq VDP$ and let VDP be such that $VDP(cfg, \cdot)$ implies that $cfg.role = I$, that all public keys in the range of $cfg.PKs_R$ are honest and that*

1. *cfg supports some version range TLS 1.x...1.3 and activates the countermeasure for all its versions,*
2. *cfg′ supports some version range TLS 1.x...1.3 and activates the countermeasure for all its versions.*

Given an adversary $\mathcal{A}$ against the downgrade security of TLS1.0-TLS1.3-sub, we construct adversaries $\mathcal{B}$ and $\mathcal{C}$ running in about the same time as $\mathcal{A}$ such that

$$\mathbf{Adv}^{\mathit{downgrade}}_{\mathit{TLS1.0\text{-}1.3\text{-}sub},\,DP'}(\mathcal{A}) \leq \mathbf{Adv}^{\mathit{version}}_{\mathit{TLS1.0\text{-}1.3\text{-}sub},\,VDP}(\mathcal{B}) + \mathbf{Adv}^{\mathit{downgrade}}_{\mathit{TLS1.3\text{-}sub},\,DP}(\mathcal{C})$$

where $DP' = DP \cup DP^{+VDP}$.

Proof sketch. Theorem 11 ensures that if the client and server configurations both support TLS 1.3 and TLS 1.2, then they get TLS 1.3 and Theorem 10 ensures that a client session that supports TLS 1.3 and TLS 1.2 gets the preferred mode when interacting with a server that only supports TLS 1.3.

The probability that the server partner session of a completing client that supports TLS 1.3 or TLS 1.3 and TLS 1.2 assigns a different version variable than 1.3 is upper bounded by $\mathbf{Adv}^{\mathsf{version}}_{\mathit{TLS1.3\text{-}1.2\text{-}sub},\,VDP}(\mathcal{B})$. Note that crucially we rely on agreement on the version variable.

The main proof idea is that servers that support TLS 1.3 and 1.2 and server that support only TLS 1.3 behave the same way *after* receiving 1.3 as the client's highest version numbers. Hence, in a first game-hop, we wait until the first Send query to a server session, and if this first query contains version number 1.3, we initialize the server with a configuration that only supports 1.3 (and besides has the configuration that the adversary provided). Else, we initialize the server with the configuration that the adversary provided. Now, all servers that run TLS 1.3 in a particular session only support TLS 1.3.

The probability that a client that supports either TLS 1.3 or TLS 1.3 and 1.2 when interacting with such a server is downgraded is upper bounded by $\mathbf{Adv}^{\mathsf{downgrade}}_{\mathit{TLS1.3\text{-}sub},\,\mathit{DP}}(\mathcal{C})$. □

8 Related Work

Downgrade as an attack vector The importance of downgrades when building practical exploits against key exchange protocols has been widely recognized [46, 12, 52, 2].

The lessons learned from this are less clear. There is a disconnect between the IETF, implementers, penetration testers, and protocol analysts. Browser developers are frequently criticized for prioritized interoperability over security and standard compliance.

`RFC 7507` [45] proposes the Signaling Cipher Suite Value (SCSV) extension for TLS to prevent version downgrade attacks when the key-exchange of all versions provides transcript authentication. SSL2.0 and SSL3.0 are being deprecated, partly to prevent version downgrade attacks as these versions do not support said extensions [50, 7] and SSL2.0 in any case does not provide reliable transcript authentication. Similarly, ciphersuite hygiene is frequently discussed in standard documents [40, 38].

Retrofitting countermeasures against downgrade attacks can inadvertently introduce or amplify attack vectors. For instance, as a countermeasure against version rollback in TLS-RSA, clients incorporate the newest protocol version they support as part of the PKCS#1-encrypted pre-master secret. Klíma et al. [34] showed that many server implementations revelead whether the version in a decrypted secret matches the version advertised in the `ClientHello` message, thus introducing a side-channel that can be exploited to implement a decryption oracle. Similar incorrect implementations of anti-Bleichenbacher countermeasures that introduce timing side-channels continue to surface in modern TLS implementations [44].

Cross protocol attacks In some cases backward compatibility or cross protocol attacks [30, 31, 6] can facilitate protocol downgrades. Our definition of downgrade security often captures such protocol weaknesses as it allows the adversary to interact with multiple instances of the protocol, each using different algorithms and potentially correlated keys and secrets.

Previous downgrade security theorems about TLS 1.2 Dowling and Stebila [24] model ciphersuite and version negotiation for the TLS protocol up to version 1.2 in the multi-ciphersuite setting introduced by Bergsma et al. [11] (discussed below). In our model, their result corresponds to a proof of downgrade security for a $\mathit{DP}(\pi.\mathit{cfg}, \pi'.\mathit{cfg})$ predicate that guarantees that all negotiable ciphersuites and versions are strong enough to provide ACCE security and that all public keys are honest and used at most by one negotiable ciphersuite. Their optimality function ω is a more limited variant of our *Nego* function and does not include entity identifiers. Their main theorem states that under such strong conditions multi-mode authentication implies downgrade security.

This is a rather weak form of downgrade security, but as shown by our attack, TLS 1.2 does not provide much stronger protection at least for clients. Servers that authenticate clients can however receive stronger guarantees.

Related work for SSH Bergsma et al. [11] previously analyzed SSH in a multi-ciphersuite setting. They split the protocol into a negotiation phase $\texttt{NP}$ and key-exchange phase $\texttt{SP}$, one for each value of $\pi.mode$. They show that if each combination $\texttt{NP}\|\texttt{SP}$ is ACCE secure, then $\texttt{NP}\|\overrightarrow{\texttt{SP}}$ is multi-ciphersuite ACCE secure. While they do not prove downgrade security per se, the result of [24] adapted to SSH corresponds to a proof of downgrade security for a $DP(\pi.cfg, \pi'.cfg)$ predicate that guarantees that all negotiable ciphersuites and versions are strong enough to provide ACCE security.

The sharing of the public key is admissible under the condition that each sub-protocol provides sufficient oracle access to the long-term key functionality, e.g., signing, to simulate all other sub-protocols. In our terminology, the protocols $\texttt{NP}\|\texttt{SP}$ of [11] are single mode restrictions of $\texttt{NP}\|\overrightarrow{\texttt{SP}}$. After their extensions with oracles providing sufficient access to long-term key functionalities, they are also sub-protocols in our sense.

We prove downgrade protection for a predicate DP that includes a much larger set of configurations. Combined with the result of Bergsma et al., our result allows to prove multi-ciphersuite ACCE security when not all sub-protocols in $\overrightarrow{\texttt{SP}}$ are ACCE secure, as long as we restrict the protocol to configurations in DP that do not negotiate them (cf. Theorem 1).

9 Conclusion and Future Work

In this paper we put forward a methodology to analyze the downgrade security of real-world key exchange protocols. Our approach breaks down the complexity of analyzing a full protocol by considering only a core sub-protocol that abstracts away details that are irrelevant for negotiation of a protocol mode. We showed that proving a simulatability property for a sub-protocol is sufficient for ensuring the *soundness* of our methodology: proving the absence of downgrade attacks on the sub-protocol is enough to guarantee the downgrade security of the full protocol. In contrast, our methodology does not provide *completeness*: it may very well be the case that a particular choice of sub-protocol abstracts too much and ends up allowing attacks that are impossible to turn into attacks on the full protocol. Indeed, sometimes sieving through false positives helped us to refine our choice of sub-protocols.

Our analysis of exemplary protocols shows that many designs fail to appropriately address downgrade security. Conversely, protocols like TLS 1.3 and SSH with strong downgrade protection follow a relatively simple design principle of authenticating all network inputs to the negotiation using a signature scheme. However, as can be observed in the Logjam like attacks on IKEv2, downgrade resilience may interact badly with other desirable security properties such as *deniability*. We recommend accounting for downgrade resilience from the outset as retrofitted downgrade protection mechanisms can be ad-hoc; see our TLS 1.2 patch. We thus advocate incorporating downgrade security as an integral part of security models for key exchange protocols.

We believe that analyzing the downgrade security of typical sub-protocols is within reach of automated tools. Symbolic analysis tools like ProVerif [19], Scyther [21] and Tamarin [43] seem particularly well suited to detect attacks on sub-protocols, helping analysts to find attacks against the full protocol or converge toward a sub-protocol that rules out false positives. Computationally sound tools like CryptoVerif [18], on the other hand, provide a means to prove the downgrade security of sub-protocols and, provided the sub-protocol is a sound abstraction of the full protocol, conclude that the corresponding full protocol also enjoys downgrade security. Finding a simulator that witnesses the correctness of a sub-protocol appears to be a more difficult task that may require ingenuity. While this may be out of reach of fully automated tools, interactive proofs can be constructed and machine-checked with tools like e.g. EasyCrypt [8]. Exploring the use of automated tools could increase the confidence in our proofs of downgrade security, and perhaps find other simpler or more practical attacks on protocols for which we only showed negative results.

Acknowledgements

We thank Cas Cremers, Travis Cross, and Antoine Delignat-Lavaud as well as the anonymous PC and student PC reviewers of S&P 2016 for their valuable comments. The TLS 1.3 downgrade countermeasures were formulated in collaboration with Eric Rescorla, Martin Thomson, and the TLS working group. Bhargavan is funded by the ERC grants CRYSP and CIRCUS. We thank Cas Cremers, Travis Cross, and Antoine Delignat-Lavaud as well as the anonymous PC and student PC reviewers of S&P 2016 for their valuable comments. Christina Brzuska is grateful to NXP for supporting her chair for IT Security Analysis

References

[1] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz. Extensible Authentication Protocol (EAP). IETF RFC 3748, 2004.

[2] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, and P. Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *22nd ACM Conference on Computer and Communications Security*, pages 5–17, 2015.

[3] N. J. AlFardan and K. G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *2013 IEEE Symposium on Security and Privacy*, pages 526–540, 2013.

[4] N. J. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. C. N. Schuldt. On the security of RC4 in TLS. In *22th USENIX Security Symposium*, pages 305–320, 2013. ISBN 978-1-931971-03-4.

[5] T. D. amd E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3, Draft 10. Internet Draft, 2008.

[6] N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J. A. Halderman, V. Dukhovni, E. K asper, S. Cohney, S. Engels, C. Paar, and Y. Shavitt. DROWN: Breaking TLS with SSLv2. https://drownattack.com/, Mar. 2016.

[7] R. Barnes, M. Thomson, A. Pironti, and A. Langley. Deprecating Secure Sockets Layer Version 3.0. IETF RFC 7568, 2015.

[8] G. Barthe, B. Grégoire, S. Heraud, and S. Zanella-Béguelin. Computer-aided security proofs for the working cryptographer. In *Advances in Cryptology, CRYPTO 2011*, pages 71–90, 2011.

[9] G. Barthe, J. M. Crespo, Y. Lakhnech, and B. Schmidt. Mind the gap: Modular machine-checked proofs of one-round key exchange protocols. In *Advances in Cryptology – EUROCRYPT 2015*, pages 689–718, 2015.

[10] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *CRYPTO 1993*, pages 232–249, 1993.

[11] F. Bergsma, B. Dowling, F. Kohlar, J. Schwenk, and D. Stebila. Multi-ciphersuite security of the secure shell (SSH) protocol. In *21st ACM Conference on Computer and Communications Security*, pages 369–381, 2014.

[12] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. Zinzindohoue. A Messy State of the Union: Taming the Composite State Machines of TLS. In *2015 IEEE Symposium on Security and Privacy*, pages 535–552, 2015.

[13] K. Bhargavan and G. Leurent. Transcript collision attacks: Breaking authentication in TLS, IKE, and SSH. In *Network and Distributed System Security Symposium (NDSS'16)*, 2016.

[14] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, and P. Strub. Implementing TLS with verified cryptographic security. In *2013 IEEE Symposium on Security and Privacy*, pages 445–459, 2013.

[15] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, P. Strub, and S. Zanella-Béguelin. Proving the TLS handshake secure (as it is). In *CRYPTO 2014*, pages 235–255, 2014.

[16] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and S. Zanella-Béguelin. Proving the tls handshake secure (as it is). Cryptology ePrint Archive, Report 2014/182, 2014. http://eprint.iacr.org/2014/182/.

[17] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright. Transport Layer Security (TLS) Extensions. IETF RFC 3546, 2003.

[18] Blanchet. A computationally sound mechanized prover for security protocols. *Dependable and Secure Computing, IEEE Transactions on*, 5(4):193–207, 2008.

[19] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *14th IEEE Computer Security Foundations Workshop, CSFW 2014*, pages 82–96,, 2001.

[20] R. Canetti and H. Krawczyk. Security analysis of IKE's signature-based key-exchange protocol. In *CRYPTO 2002*, pages 143–161, 2002.

[21] C. Cremers. The Scyther tool: Verification, falsification, and analysis of security protocols. In *20th International Conference on Computer Aided Verification, CAV 2008*, pages 414–418, 2008.

[22] C. J. F. Cremers. Key exchange in IPsec revisited: Formal analysis of IKEv1 and IKEv2. In *16th European Symposium on Research in Computer Security – ESORICS 2011*, pages 315–334, 2011.

[23] A. Delignat-Lavaud and K. Bhargavan. Network-based origin confusion attacks against HTTPS virtual hosting. In *24th International Conference on World Wide Web, WWW 2015*, pages 227–237, 2015.

[24] B. Dowling and D. Stebila. Modelling ciphersuite and version negotiation in the TLS protocol. In *20th Australasian Conference on Information Security and Privacy, ACISP 2015*, pages 270–288, 2015. doi: 10.1007/978-3-319-19962-7_16.

[25] B. Dowling, M. Fischlin, F. Günther, and D. Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In *22nd ACM Conference on Computer and Communications Security*, pages 1197–1210, 2015.

[26] T. Duong and J. Rizzo. Here come the $\oplus$ ninjas. Available at http://nerdoholic.org/uploads/dergln/beast_part2/ssl_jun21.pdf, May 2011.

[27] C. Garman, K. G. Paterson, and T. V. der Merwe. Attacks only get better: Password recovery attacks against RC4 in TLS. In *24th USENIX Security Symposium*, pages 113–128, 2015. ISBN 978-1-931971-232.

[28] D. Harkins and D. Carrel. The internet key exchange (IKE). IETF RFC 2409, 1998. URL http://www.ietf.org/rfc/rfc2409.txt.

[29] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk. On the security of TLS-DHE in the standard model. In *CRYPTO 2012*, pages 273–293, 2012.

[30] T. Jager, K. G. Paterson, and J. Somorovsky. One bad apple: Backwards compatibility attacks on state-of-the-art cryptography. In *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013*, 2013. URL http://internetsociety.org/doc/one-bad-apple-backwards-compatibility-attacks-state-art-cryptography.

[31] T. Jager, J. Schwenk, and J. Somorovsky. On the security of TLS 1.3 and QUIC against weaknesses in PKCS#1 v1.5 encryption. In *22nd ACM Conference on Computer and Communications Security*, pages 1185–1196, 2015.

[32] M. Just and S. Vaudenay. Authenticated multi-party key agreement. In *ASIACRYPT 1996*, pages 36–49, 1996.

[33] C. Kaufman, P. Hoffman, Y. Nir, and P. Eronen. Internet Key Exchange Protocol Version 2 (IKEv2). IETF RFC 5996, 2010.

[34] V. Klíma, O. Pokorný, and T. Rosa. Attacking RSA-based sessions in SSL/TLS. In *5th International Workshop on Cryptographic Hardware and Embedded Systems – CHES 2003*, pages 426–440. Springer, 2003.

[35] K. Kobara, S. Shin, and M. Strefler. Partnership in key exchange protocols. In *2009 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2009*, pages 161–170, 2009.

[36] H. Krawczyk. SIGMA: the 'SIGn-and-MAc' approach to authenticated Diffie-Hellman and its use in the IKE protocols. In *CRYPTO 2003*, pages 400–425, 2003.

[37] H. Krawczyk, K. G. Paterson, and H. Wee. On the security of the TLS protocol: A systematic analysis. In *CRYPTO 2013*, pages 429–448, 2013.

[38] A. Langley, N. Modadugu, and B. Moeller. Transport Layer Security (TLS) False Start. Internet Draft, 2010.

[39] G. Lowe. A hierarchy of authentication specification. In *10th Computer Security Foundations Workshop (CSFW '97)*, pages 31–44. IEEE Computer Society, 1997.

[40] R. H. M. Salter, E. Rescorla. Suite B Profile for Transport Layer Security (TLS). IETF RFC 5430, 2009.

[41] N. Mavrogiannopoulos, F. Vercauteren, V. Velichkov, and B. Preneel. A cross-protocol attack on the TLS protocol. In *19th ACM Conference on Computer and Communications Security*, pages 62–72, 2012. ISBN 978-1-4503-1651-4. doi: 10.1145/2382196.2382206.

[42] C. Meadows. Analysis of the internet key exchange protocol using the NRL protocol analyzer. In *1999 IEEE Symposium on Security and Privacy*, pages 216–231, 1999.

[43] S. Meier, B. Schmidt, C. Cremers, and D. Basin. The tamarin prover for the symbolic analysis of security protocols. In *25th International Conference on Computer Aided Verification, CAV 2013*, pages 696–701, 2013.

[44] C. Meyer, J. Somorovsky, E. Weiss, J. Schwenk, S. Schinzel, and E. Tews. Revisiting ssl/tls implementations: New bleichenbacher side channels and attacks. In *23rd USENIX Security Symposium*, pages 733–748. USENIX Association, 2014.

[45] B. Moeller and A. Langley. TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks. IETF RFC 7507, 2015.

[46] B. Möller, T. Duong, and K. Kotowicz. This POODLE Bites: Exploiting The SSL 3.0 Fallback. Available at https://www.openssl.org/~bodo/ssl-poodle.pdf, 2014.

[47] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3, Draft 10. Internet Draft, 2015.

[48] J. Rizzo and T. Duong. The CRIME Attack, September 2012.

[49] P. Rogaway and T. Stegers. Authentication without elision: Partially specified protocols, associated data, and cryptographic models described by code. In *Proceedings of the 22nd IEEE Computer Security Foundations Symposium, CSF 2009, Port Jefferson, New York, USA, July 8-10, 2009*, pages 26–39, 2009.

[50] S. Turner and T. Polk. Prohibiting Secure Sockets Layer (SSL) Version 2.0. IETF RFC 6176, 2011.

[51] University of Michigan. Tracking the FREAK Attack. Available at https://freakattack.com/, November 2015.

[52] D. Wagner and B. Schneier. Analysis of the SSL 3.0 protocol. In *2nd USENIX Workshop on Electronic Commerce, WOEC 1996*, pages 29–40, 1996.

[53] T. Ylonen and C. Lonvick. The secure shell (SSH) authentication protocol. IETF RFC 4252, 2006.

[54] T. Ylonen and C. Lonvick. The secure shell (SSH) transport layer protocol. IETF RFC 4253, 2006.

[55] P. Zimmermann. RFC 6189bis: ZRTP: Media Path Key Agreement for Unicast Secure RTP, 2012.