Downgrade Resilience in Key-Exchange Protocols

Karthikeyan Bhargavan*, Christina Brzuska†, Cédric Fournet‡, Matthew Green§,
Markulf Kohlweiss‡ and Santiago Zanella-Béguelin‡
*Inria Paris-Rocquencourt, Email: karthikeyan.bhargavan@inria.fr
†Hamburg University of Technology, Email: brzuska@tuhh.de
‡Microsoft Research, Email: {fournet,markulf,santiago}@microsoft.com
§Johns Hopkins University, Email: mgreen@cs.jhu.de

***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.

I. 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 [2, 3, 4, 10, 23], 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 [39].

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 [43] and even in recent versions of TLS [2, 39].

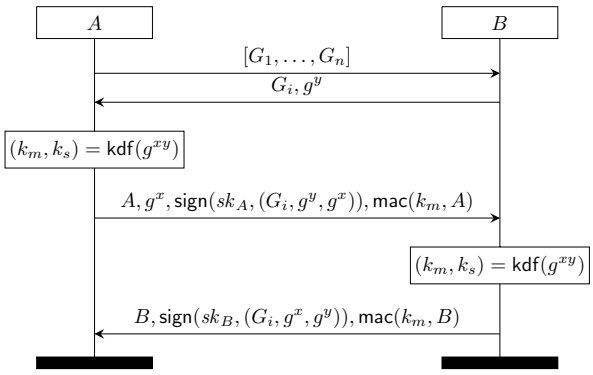


Fig. 1: SIGMA-N: Basic SIGMA [30] with group negotiation

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 [25, 31] only model a single mode at a time. Some [12, 13] do model negotiation of *weak* algorithms, but do not guarantee negotiation of the preferred mode. Some others [9, 21] 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. [2, 10, 43], and because recent Internet-wide scans have revealed the prevalence of hosts supporting insecure protocol modes [2, 5, 42].

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

A. Motivating example

We begin with a simple motivating example: we adapt the SIGMA protocol of Krawczyk [30] 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

© 2016, Karthikeyan Bhargavan. Under license to IEEE.
DOI 10.1109/SP.2016.37

IEEE computer society

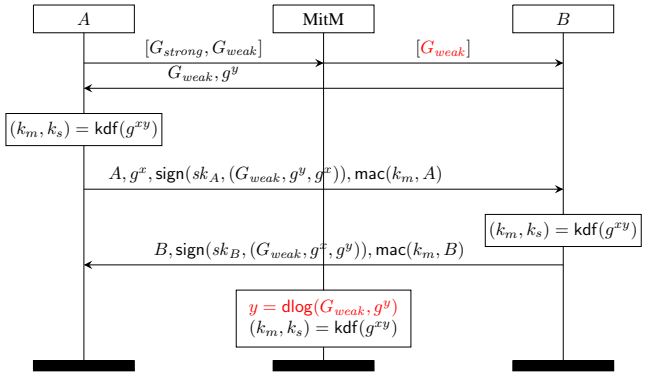


Fig. 2: Man-in-the-Middle downgrade attack on SIGMA-N

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. 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 [32], which is being codified as part of TLS 1.3 [40].

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 [2, 35]. 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.

B. 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. [13], 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 [13] 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 in the MITLS framework, two parties under adversarial influence may arrive at an 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 peer configurations should 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 just one possible mode are trivially 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 technique of *lifting* security from the sub-protocol to the main protocol was previously employed by Bergsma et al. [9] to prove multi-ciphersuite security.

In our analysis we manually extract sub-protocols that 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 of Bergsma et al. [9]. 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.

C. 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, as well as sufficient conditions under which these protocols achieve downgrade security. These conditions inform practitioners 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 vulnerabilities are present in the protocol standards, but 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 [40] 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.

D. Outline of the paper

In §II we introduce the terminology used throughout and we provide a primer on security definitions for key exchange protocols. We formally define downgrade resilience in §III. In §IV through §VII we apply these definitions to analyze the security of SSH, IPSec IKE, ZRTP and TLS. We survey related work in §VIII and conclude in §IX. The full version [14] includes additional discussions and proofs.

II. Modeling Multi-Mode Key-Exchanges

Popular key-exchange models [9, 13, 22] focus mainly on entity authentication and key-indistinguishability [8]. Our focus lies on considering multi-mode protocols and incorporating the negotiation of the mode into the security model.

A key exchange protocol Π is a two-party protocol with an initiator role I and a responder role R (sometimes called client and server). The adversary interacts with multiple sessions of the protocol. 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:

$\pi.\mathit{cfg}$	initial configuration (including the role);
$\pi.\mathit{uid}$	unique identifier of the session;
$\pi.\mathit{mode}$	negotiated *mode* (including long-term identities);
$\pi.\mathit{key}$	session key;
$\pi.\mathit{complete}$	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 $\pi.\mathit{cfg}$ is assigned when a session is created and contains other variables, including one for the session role. We use $\pi.\mathit{role}$ as shorthand for $\pi.\mathit{cfg}.\mathit{role}$ and let $\overline{I} = R$, $\overline{R} = I$.

An adversary interacts with sessions via queries to oracles. A query $\pi \leftarrow \mathsf{Init}(\mathit{cfg})$ 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 $m_{\mathit{out}} \leftarrow \mathsf{Send}(\pi, m_{\mathit{in}})$ 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 $\pi.\mathit{key}$. There are several variants of this setting for handling long-term keys and other authentication mechanisms as well as corruption settings, and each variant requires different variables and oracles. As these settings are mostly standard and orthogonal to our definition, we leave those details unspecified for now and get back to them in Section IV. Note that our definitions only become complete once we add the specifics of long-term keys or other authentication mechanisms.

A. Unique identifiers and partnering

The goal of a key exchange protocol is to establish sessions between two different parties so that they compute the same key and agree on the algorithms and authentication setting. We say that two sessions match when they derive the same session key [29]. For defining downgrade resilience, we rely on the weaker notion of *partnering*, based on unique identifiers—*at most* two sessions may assign the same value to *uid*.

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

For example, in TLS, a suitable value for *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 *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 2 (Partnering security): The advantage of adversary $\mathcal{A}$ against the partnering security of Π, $\mathbf{Adv}^{\mathsf{partnering}}_{\Pi,\,PS}(\mathcal{A})$, is the probability that, when $\mathcal{A}$ interacts with protocol Π, there is an unpartnered session π such that $\pi.\mathit{complete} = \mathsf{true}$ and $\mathit{PS}(\pi.\mathit{cfg})$ holds.

That is, partnering security requires that sessions that complete with a protected configuration have at least one partner session that assigns the same value to *uid*.

B. 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* [18] and the type of authentication (e.g. mutual or bilateral) is determined as part of the negotiation [22]. 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 $\mathit{mode}.\mathit{eid}_r$. 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 $\mathit{Auth}(\mathit{mode}, r)$, which holds when *mode* is expected to authenticate role $\bar{r}$. Authentication as defined by Lowe [33] guarantees agreement on the variables of authenticated peers.

Definition 3 (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 $x \in X$.

When defining downgrade resilience, we consider a weaker notion that guarantees equality once both peers have assigned a value to x.

Definition 4 (Pre-agreement): A session π pre-agrees with π' on x when $\pi.x = \pi'.x$ or $\pi'.x = \bot$.

We consider authentication with agreement. Authentication with pre-agreement can be defined analogously.

Definition 5 (Multi-mode authentication): A session π completes maliciously for X when $\pi.\mathit{complete} = \mathsf{true}$ but there is no partnered session π' matching π that agrees with π on X.

The advantage $\mathbf{Adv}^{\mathsf{mm\text{-}auth}}_{\Pi,\,\mathit{Auth},\,X}(\mathcal{A})$ 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 $\mathit{Auth}(\pi.\mathit{mode}, \pi.\mathit{role})$ holds.

Let $r = \pi.\mathit{role}$. Note that $\mathit{Auth}(\pi.\mathit{mode}, r)$ typically includes the requirement that the long term key $\pi.\mathit{eid}_{\bar{r}}$ of the peer is honest. If, as in SIGMA-N, the mode is secure against *key-compromise impersonation* attacks [26] then $\pi.\mathit{eid}_r$ need not be honest. In addition, the predicate *Auth* models concurrent mixed-mode authentication. A protocol *mode* provides mutual-authentication if $\mathit{Auth}(\pi.\mathit{mode}, r)$ holds regardless of r. It provides server-only authentication if only $\mathit{Auth}(\pi.\mathit{mode}, I)$ 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 [20]. As we will see, protocols need to have sufficient downgrade resilience to guarantee that the preferred authentication mode is negotiated.

C. 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.

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

Identity protection and *deniability* are other orthogonal security requirements of key-exchange protocols. Although we do not formally present them here, we note that many design decisions in real-world key-exchange protocols are motivated by user privacy in addition to the more common security goals of key indistinguishability and entity authentication.

D. 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

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

III. Defining Downgrade Resilience

Downgrade resilience is motivated by protocols such as SIGMA-N that despite satisfying 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 to the protocol mode negotiated (if

any) in the absence of active adversaries. Formally, if a session π with role r talking to a session π' completes, it must be the case that $\pi.mode = Nego_r(\pi.cfg, \pi'.cfg)$, where $Nego_r$ is an abbreviation defined by case:

$$Nego_r(cfg_r, cfg_{\bar{r}}) \triangleq \begin{cases} Nego(cfg_r, cfg_{\bar{r}}) & \text{when } r = I \\ Nego(cfg_{\bar{r}}, cfg_r) & \text{when } r = R\ . \end{cases}$$

Definition 6 (Negotiation correctness): The protocol negotiation is correct if, whenever a session π with role r and configuration cfg_r completes, there exists a peer configuration $cfg_{\bar{r}}$ such that $\pi.mode = Nego_r(cfg_r, cfg_{\bar{r}})$.

This property captures that, if a protocol mode is disabled by 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 **D**owngrade **P**rotection predicate *DP* on pairs of configurations.

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

Definition 7 (Downgrade security): A session π is downgraded when $\pi.complete = \mathsf{true}$ and there is a partnered session π' such that *DP*$(\pi.cfg, \pi'.cfg)$ holds, but $\pi.mode \neq Nego_{\pi.role}(\pi.cfg, \pi'.cfg)$.

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 is 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 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 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.

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

The *DP* predicate for downgrade protection plays a role similar to *Auth* for authentication, except 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 influenced by an adversary.

Our formal configurations are session-specific, and do not necessarily coincide with concrete configurations 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 with different long-term keys, for instance, using the server name indication extension [15].

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. In this case the only influence $\pi'.cfg$ has on the downgrade protection of π is in the level of agility, i.e. will π' use its long-term keys also in other protocol and cipher modes? 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.

A. Downgrade resilience of SIGMA-N

Recall that SIGMA-N configurations are tuples of the form $(r, ID, pk_{ID}, PK, groups)$ 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, $Nego(cfg_I, cfg_R)$ is defined as

$$(nego(cfg_I.groups, cfg_R.groups), cfg_I.pk_A, cfg_R.pk_B)$$

when $cfg_I.pk_A = cfg_R.PK(A)$, and $\perp$ otherwise.

For such a *Nego* function, *DP* can hold only for pairs of configurations with at most one group in common, as shown by the attack in Fig. 2.

B. 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 universe of 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 8 (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 security): 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

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

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 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 the experiment).

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 $\mathbf{Adv}^{\mathsf{auth}}_{\Pi, Auth, X}(\mathcal{A})$. ■

Interestingly, partnering security is similar to the aliveness requirement in some (single-mode) security definitions which Krawczyk [30] 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.

We sketch a similar theorem for key-indistinguishability in the full version [14].

C. Downgrade secure sub-protocols

We are interested in minimal core sub-protocols that guarantee downgrade security. We justify our use of sub-protocols as a sound abstraction of a full protocol using simulation. A sub-protocol can take additional input as part of Init and Send queries to allow for an accurate simulation of the execution of the full protocol. This is akin to the sub-protocols of Bergsma et al. [9] 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 9 (Sub-protocol): A protocol $\widetilde{\Pi}$ is a sub-protocol of Π for X if we have an efficient simulator $\mathcal{S}$ with access to the oracles of $\widetilde{\Pi}$ such that:

1) $\mathcal{S}$ transparently relays queries to oracles for reading variables in X to the same oracles in $\widetilde{\Pi}$.

2) $\mathcal{S} \circ \widetilde{\Pi}$ is information-theoretic indistinguishable from Π.

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 $\widetilde{\Pi}$. That is, $\mathcal{S}$ may use 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 $\{cfg, uid, mode, key, complete\} \cup X$, and *DP* a downgrade protection predicate. Let $\mathcal{S}$ be a simulator for $\widetilde{\Pi}$ as in Definition 9. Then, for any adversary $\mathcal{A}$ against the downgrade security of Π with pre-agreement on X, $\mathcal{A} \circ \mathcal{S}$ is an adversary against the downgrade security of $\widetilde{\Pi}$ with pre-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 occurring 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. ∎

An analogous theorem holds for partnering security.

D. Downgrade security by whitelisting

Consider a protocol that is negotiation correct and guarantees multi-mode authentication with pre-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 [21].

IV. Secure SHell

Figure 3a models a run of the SSHv2 [45] protocol with a client that authenticates using the `publickey` method [44].

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 `publickey`.

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 [45, 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 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 II, the sub-protocol uses the following session variables:

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

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 sets *mode* to:

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

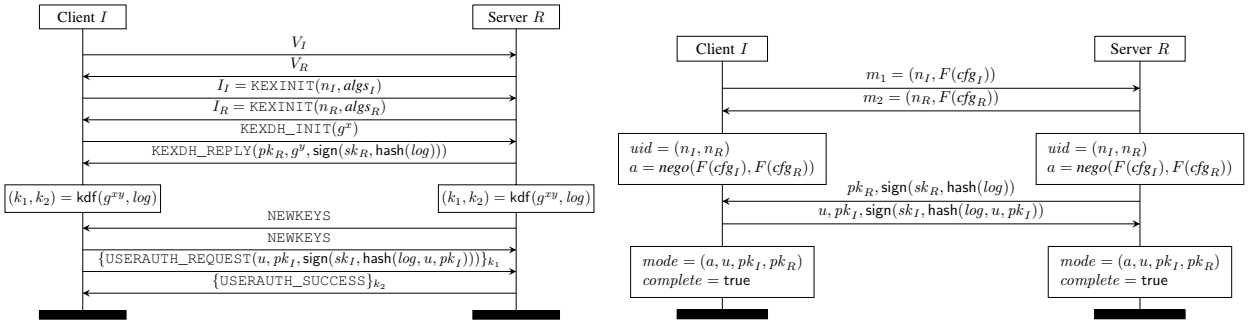
The server's behavior is specified analogously.

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 3 (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. ∎



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

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

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

Agile hash functions and signatures

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

Definition 10 (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. [13] 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 11 (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 *SSH-sub*. By virtue of Theorem 2 it suffices to prove that *SSH-sub* is downgrade secure. The same reasoning applies to partnering security.

Partnering security

Define $Nego(cfg_I, cfg_R) \triangleq (a, cfg_I.u, cfg_I.PK_I(a,u), pk_R)$, where $a = nego(F(cfg_I), F(cfg_R))$ if $pk_R = cfg_R.PK_R(a)$, and $\perp$ otherwise. Let $\mathcal{M}$ be the set of all supported modes and $\mathcal{H}$ be the set of all supported hash algorithms. We define

$$\mathcal{M}^\star \triangleq \{Nego_{cfg.role}(cfg, cfg') | PS(cfg)\}$$
$$\mathcal{H}^\star \triangleq \{mode.hash \mid mode \in \mathcal{M}^\star\}$$
$$\mathcal{P}_s \triangleq \{p \mid s, p = mode.\overline{sig} \wedge mode \in \mathcal{M}\}$$

That is, $\mathcal{M}^\star$ are the modes negotiated by pairs of configurations where the first configuration guarantees partnering security, $\mathcal{H}$ are the hash algorithms used in partnering secure modes, and $\mathcal{P}_s$ are the agility parameters for the peer signature scheme s.

Theorem 4 (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 $\mathcal{B}_{s,p}$ 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

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

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

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 5 (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 $\mathcal{B}_{s,p}$ and $\mathcal{B}_h$ running in about the same time as $\mathcal{A}$ such that $\mathbf{Adv}^{\mathsf{downgrade}}_{\textit{SSH-sub}, DP}(\mathcal{A})$ is at most

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

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

V. 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 [24] and IKEv2 [27]. Both variants are inspired by the SIGMA protocol [30] recalled in the introduction, and are believed to inherit its authentication and key-indistinguishability guarantees. Next, we study their downgrade protection sub-protocols.

A. 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 corresponding downgrade protection sub-protocol is depicted in Figure 4b.

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 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 $\mathtt{SA}_R$. In practice, they are effectively HMAC-MD5 or HMAC-SHA1. For simplicity, in the following we assume that configurations specify fixed kdf and mac algorithms. (See §IV for an explicit handling of cryptographic agility.) We use the following notations for the sub-protocol:

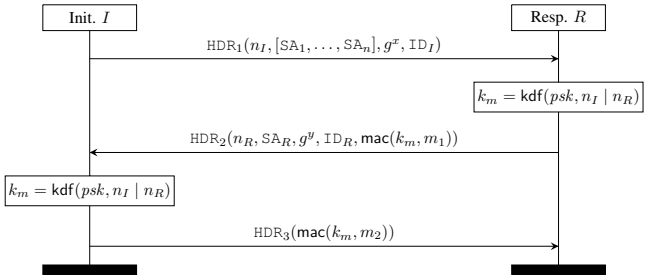
- the goal is to agree on a mode $(\mathtt{SA}_R, ID_I, ID_R)$;
- $cfg_I = (ID_I, [\mathtt{SA}_1, \ldots, \mathtt{SA}_n])$.
- cfg_R includes ID_R and is otherwise unspecified; it would typically also include a list of SAs.
- F is a formatting function from cfg_I to the payload of the first message that 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.

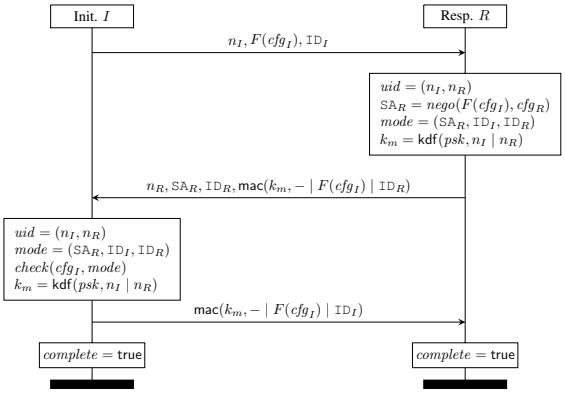
We also assume that the protocol rejects runs in which $ID_I = ID_R$. This is referred to in the literature as the *self-communication* scenario, and in such settings there are well known reflection attacks on IKEv1 [19, 36]. Positive results in this setting would require an extension of our sub-protocol and assumptions about the DH groups or the ISAKMP cookies employed in the protocol.

We first prove partnering security (Definition 2) 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).

Theorem 6 (Partnering security of IKEv1 sub-protocol): 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



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



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

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

about the same time as $\mathcal{A}$ such that $\mathbf{Adv}^{\mathsf{partnering}}_{\textit{IKEv1-sub, PS}}(\mathcal{A})$ 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 *psk*s employed by sessions.

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.

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

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.

B. IKEv2 does not prevent downgrade attacks

IKEv2 [27] is a revision of the IKEv1 protocol intended to simplify the specification and extend it to cover popular authentication methods such as EAP [1].

IKEv2 with signatures: We first consider the plain, signature-based protocol and sub-protocol in Figures 5a and 5b.

We 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.

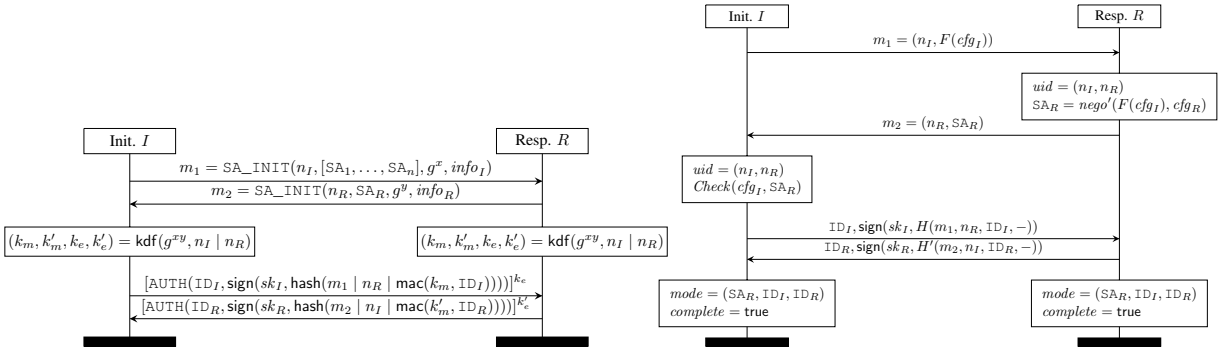
As in IKEv1, the initiator begins by offering a sequence of security associations (extracted from cfg_I) and the responder chooses one of these. In the full protocol, the initiator and responder also exchange Diffie-Hellman public values and use them to derive session keys, used (in particular) to encrypt 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 $\mathtt{SA}_R$ 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 a weak group. The attack proceeds as follows (see Fig. 6). 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 to the initiator, who thinks that the responder only supports group 1. If the attacker has performed enough pre-computation so as to be able to compute discrete logs in group 1, then he can compute the session and MAC keys and impersonate the responder.

In practice, executing this attack requires the MitM to send



(a) IKEv2 protocol with mutual signatures.

(b) IKEv2 sub-protocol with mutual signatures.

Fig. 5: IKEv2 protocol and sub-protocol for signature-based authentication

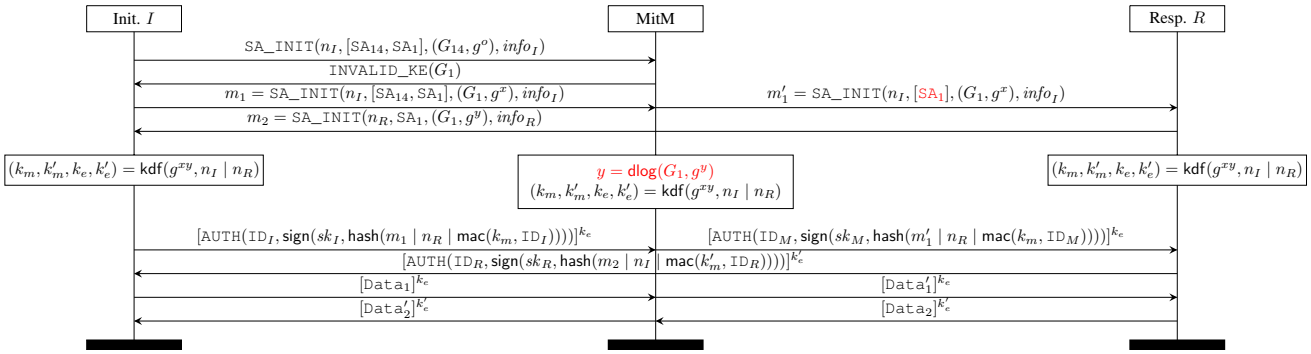


Fig. 6: Man-in-the-middle downgrade on IKEv2 with mutual signatures and weak Diffie-Hellman groups

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.

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 Figures 7a and 7b).

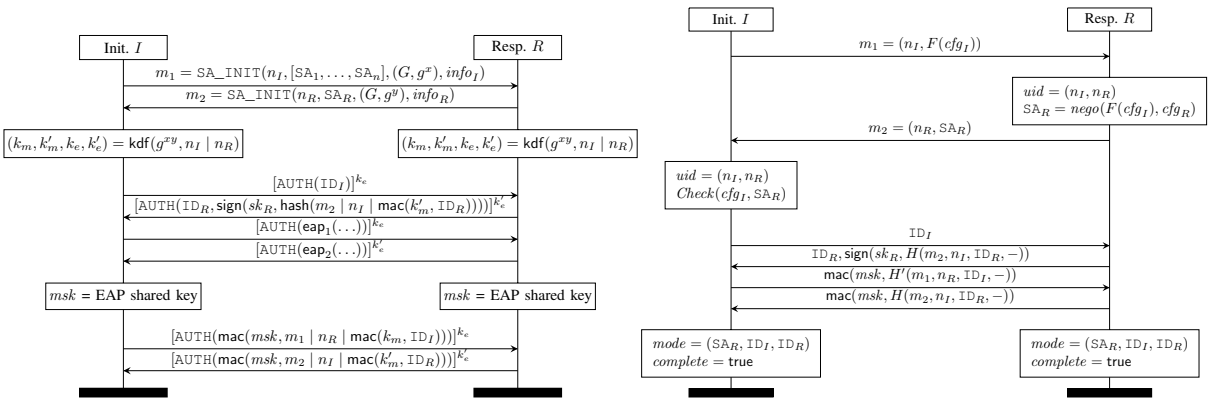
In this variant, in the third message, the initiator sends its identity without any signature. Instead, after verifying the server's signature, it engages in an application-level 'embedded' authentication protocol that generates a shared key. Its use of EAP is asymmetric, in that EAP authenticates the 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



(a) IKEv2 protocol with EAP client authentication.

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

Fig. 7: IKEv2 protocol and sub-protocol for EAP-based authentication

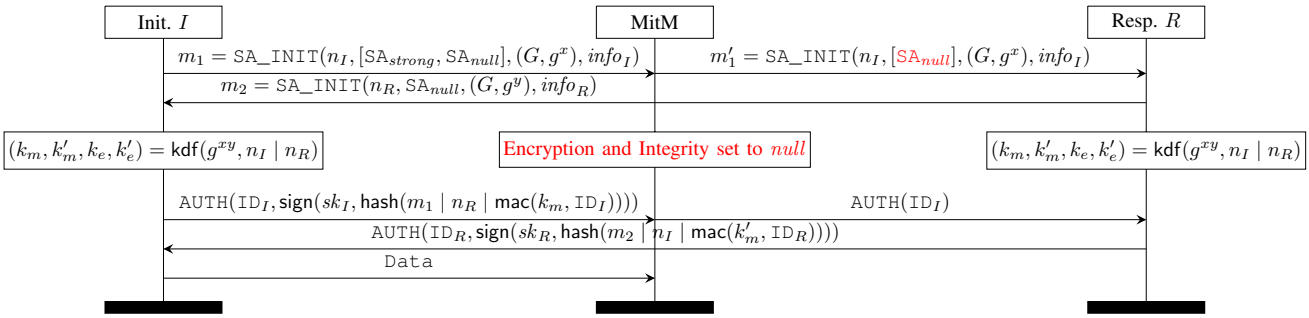


Fig. 8: Man-in-the-middle cross-protocol downgrade on IKEv2 mixing signatures and EAP authentication

algorithm or ENCR_NULL as the IKE encryption algorithm." [27, 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.

C. 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" [27, Section 2.5].

VI. Z Real-Time Protocol

ZRTP [46] 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.

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.

A. 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 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 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. The attack begins with the establishment of a shared key (via Diffie-Hellman) before restarting with the PSK mode. We describe the attack in detail in the full version [14]. In practice, most ZRTP implementations do not implement pre-shared mode, and those that do only allow SAS authentication after DH exchanges. Nevertheless, this protocol-level attack should serve as a cautionary tale for future ZRTP implementations and extensions.

VII. 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 2.0 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 [43] and a dangerous backward compatibility attack [5]. 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 [10] and Logjam [2].

Since SSL 3.0, 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).

A. Negotiation in TLS 1.0–1.2

Figure 12a 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

[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.

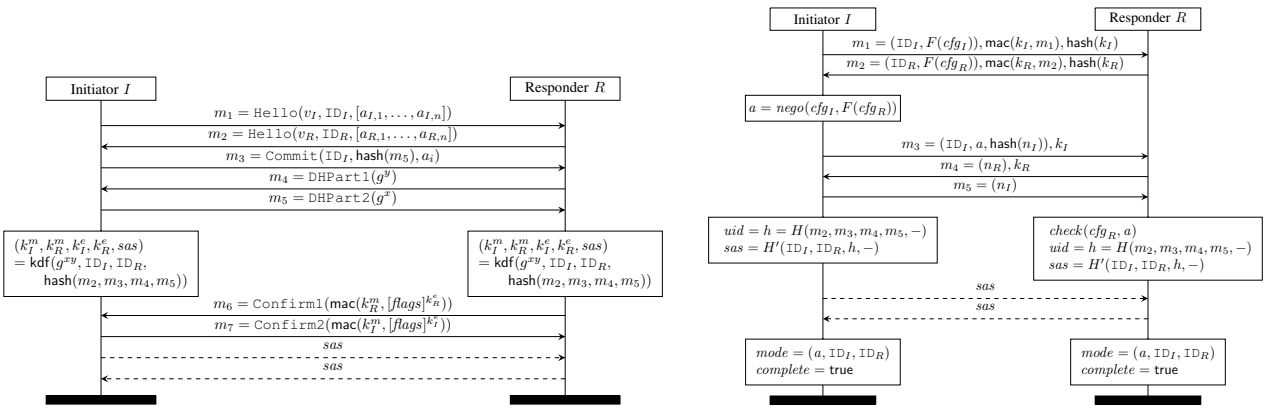


Fig. 9: ZRTP (a) protocol (b) downgrade protection sub-protocol

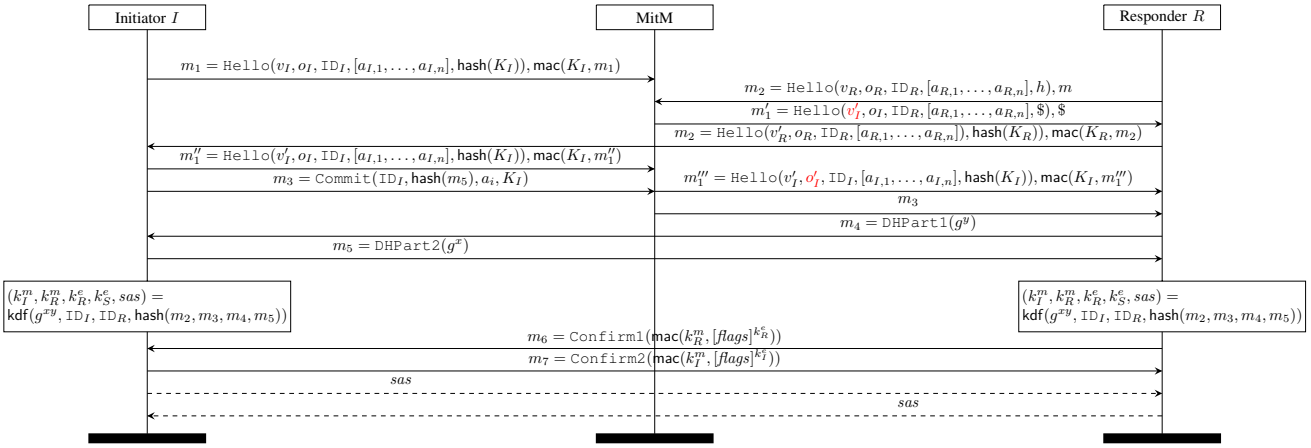


Fig. 10: Man-in-the-Middle attack on ZRTP version and option negotiation. We assume that both peers prefer version $v_I = v_R$, but will support an older version $v_I' = v_R'$. The attacker additionally modifies the options flags o_I transmitted in m_1.

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 ($[\texttt{Data}]^k$).

B. TLS 1.0–1.2 do not prevent downgrades

The downgrade protection sub-protocol for TLS 1.0–1.2 is depicted in Fig. 12b. 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 messages (mac, 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

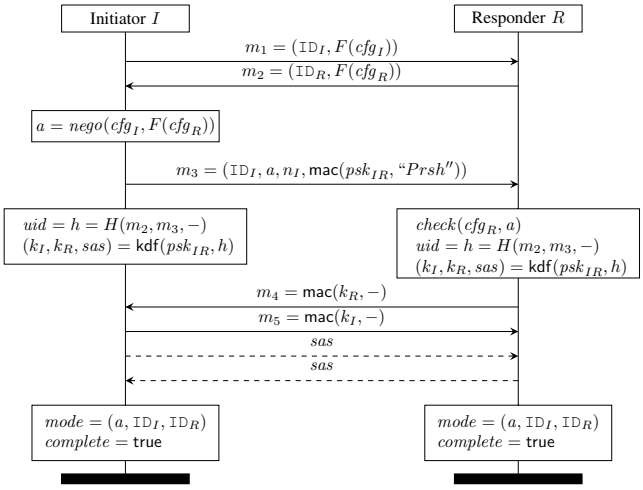


Fig. 11: ZRTP with Pre-Shared Keys: negotiation sub-protocol

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 [11].

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 II, the downgrade protection sub-protocol uses the following session 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 [35]) 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 [21] 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 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.

C. On downgrade protection in Draft 10 of TLS 1.3

Draft 10 of TLS 1.3 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. 13a. The corresponding downgrade protection sub-protocol is in Fig. 13b.

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

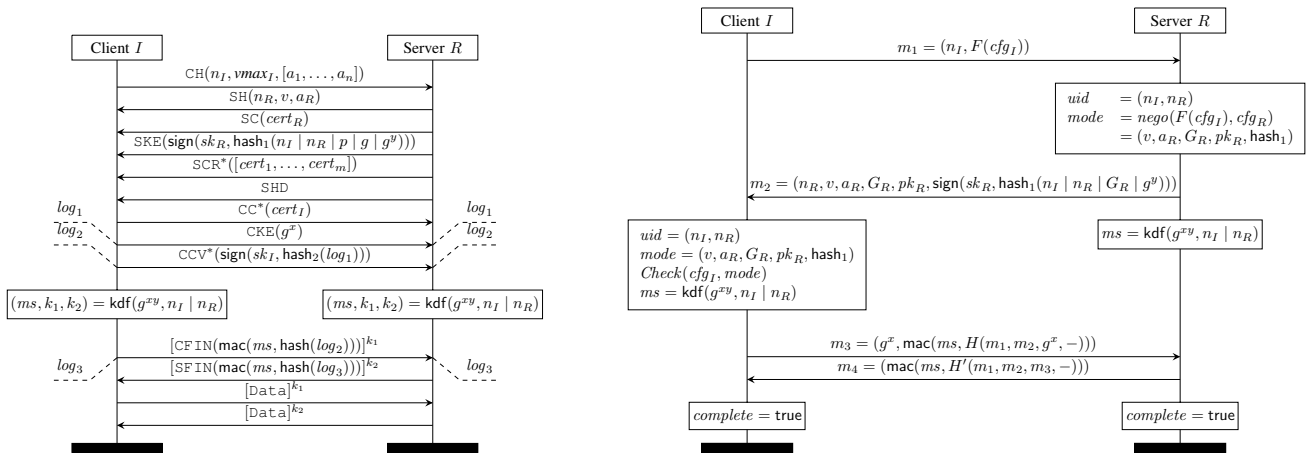


Fig. 12: TLS 1.0–1.2 with (EC)DHE key exchange (a), where messages labeled with * occur only when client authentication is enabled, and (b) its downgrade protection sub-protocol

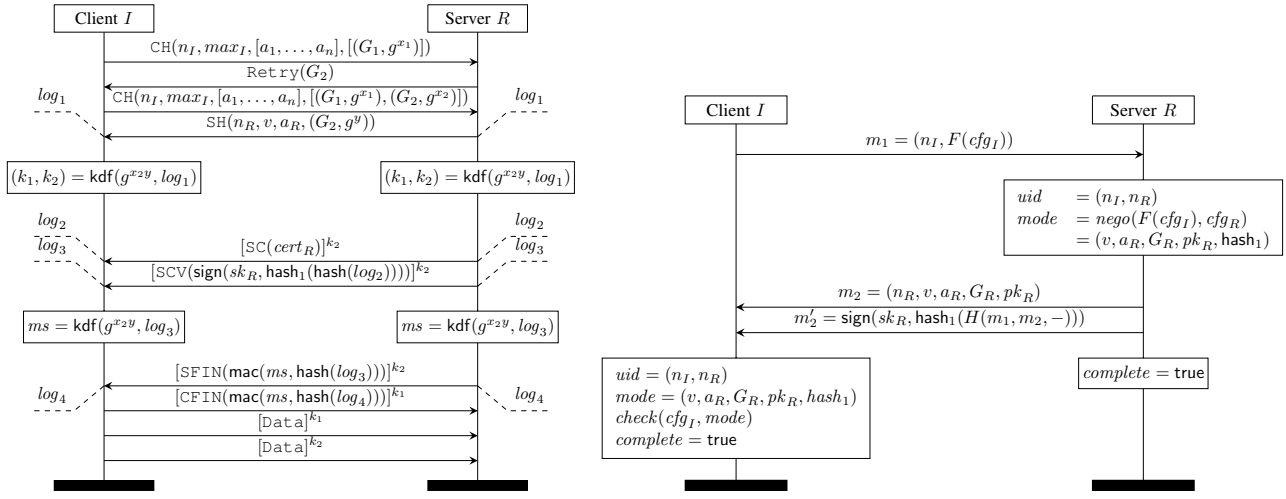


Fig. 13: TLS 1.3 1-RTT mode with server-only authentication (a) and its downgrade protection sub-protocol (b)

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 [38], 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. 14a. 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. 14a. 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 and downgrade security for clients speaking to servers that implement the fix described in Fig. 14a. We then define version downgrade security and show that the fixes in Fig. 14a (TLS 1.3) and Fig. 14b (TLS 1.0–1.2) prevent version downgrade.

In all of our theorems we consider a universe of configurations that support subsets of TLS 1.0, 1.1, 1.2 and 1.3 and that enable only (EC)DHE. Note that RSA keys are thus used for signing only.

Theorem 7 (Partnering security of TLS 1.0–1.3): Let *PS* be such that $\mathit{PS}(\mathit{cfg})$ implies that $\mathit{cfg.role} = I$ and that all public keys in the range of $\mathit{cfg.PKs}_R$ are honest. Given an adversary $\mathcal{A}$ against the partnering security of our sub-protocol, we construct adversaries $\mathcal{B}_{s,p}$ and $\mathcal{B}_h$ running in about the same time as $\mathcal{A}$ such that $\mathbf{Adv}^{\mathsf{partnering}}_{\mathit{TLS1.0\text{-}1.3\text{-}sub},\, \mathit{PS}}(\mathcal{A})$ is at most

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

where n_s keys are generated for signing scheme s.

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., $\mathcal{M}^\star = \{\mathit{Nego}_{\mathit{cfg.role}}(\mathit{cfg}, \mathit{cfg}') \mid \mathit{DS}(\mathit{cfg}, \mathit{cfg}')\}$ and $\mathcal{H}^\star = \{\mathit{mode.hash} \mid \mathit{mode} \in \mathcal{M}^\star\}$.

Theorem 8 (Downgrade security of TLS 1.3): Let *DP* be such that $\mathit{DP}(\mathit{cfg}, \mathit{cfg}')$ implies that $\mathit{cfg.role} = I$, that all public keys in the range of $\mathit{cfg.PKs}_R$ are honest and such that: (a) *cfg* supports at least TLS 1.3 and implements the countermeasure. (b) *cfg*′ only supports TLS 1.3 and implements the countermeasure. Given an adversary $\mathcal{A}$ against the downgrade security of TLS 1.3 sub-protocol, we construct adversaries $\mathcal{B}_{s,p}$ and $\mathcal{B}_h$ running in about the same time as $\mathcal{A}$ such that $\mathbf{Adv}^{\mathsf{downgrade}}_{\mathit{TLS1.3\text{-}sub},\, \mathit{DP}}(\mathcal{A})$ is at most

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

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.)

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 12 (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 $\pi.v \neq \mathit{Version}_{\pi.\mathit{role}}(\pi.\mathit{cfg}, \pi'.\mathit{cfg})$.

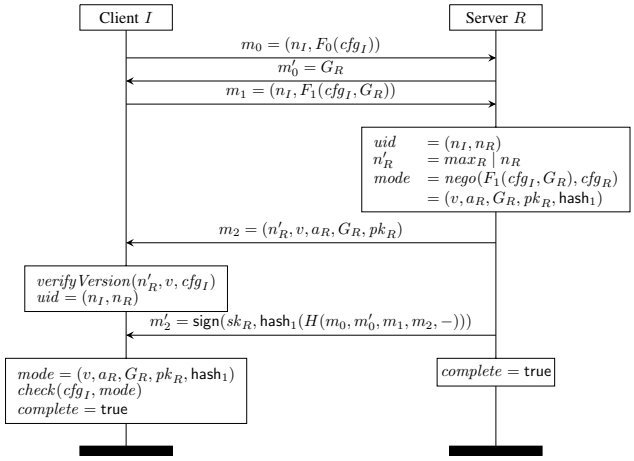
Theorem 9 (Version downgrade security of TLS 1.0–1.3): Let *VDP* be such that $\mathit{VDP}(\mathit{cfg}, \mathit{cfg}')$ implies that $\mathit{cfg.role} = I$ and that all public keys in the range of $\mathit{cfg.PKs}_R$ are honest and such that both *cfg* and *cfg*′ support at least TLS 1.3 and activate the countermeasure.

Given an adversary $\mathcal{A}$ against the version downgrade security of our sub-protocol, we construct adversaries $\mathcal{B}_{s,p}$ 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

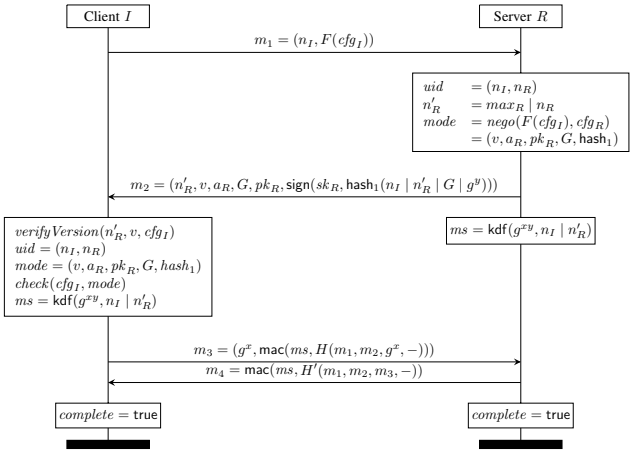
$$\frac{n^2}{2^{|n_R|+1}} + \sum_{h \in \mathcal{H}^\star} \mathbf{Adv}^{\mathsf{CR}}_{h,\,\mathcal{H}}(\mathcal{B}_h) + \sum_{(s,p) \in \overline{\mathit{sig}}(\mathcal{M}^\star)} n_s \mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}_{s,\,p,\,\mathcal{P}_s}(\mathcal{B}_{s,p})\,,$$

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.

For predicates *DP* and *VDP* such that $\mathit{DP} \subseteq \mathit{VDP}$, let $\mathit{DP}^{+\mathit{VDP}}$ be the predicate that holds for pairs of configurations in *DP*, with server configurations extended to support configurations of lower version protocols that by *VDP* should never be negotiated. Putting Theorems 8 and 9 together, we get that when both client and server implement the countermeasures, then clients supporting multiple versions including TLS 1.3 are



(a) A version downgrade fix for TLS 1.3



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

as good as supporting only TLS 1.3. Version 1.3 needs to be always present as our version downgrade guarantees concern downgrades from 1.3 to lower versions.

Corollary 1 (Downgrade security of TLS 1.0–1.3): Assume $DP \subseteq VDP$ 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 *cfg* and cfg' support at least TLS 1.3 and activate the countermeasure. Given an adversary $\mathcal{A}$ against the downgrade security of our sub-protocol, we construct adversaries $\mathcal{B}$ and $\mathcal{C}$ running in about the same time as $\mathcal{A}$ such that

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

VIII. Related Work

Downgrade as an attack vector: The role of downgrade attacks in practical exploits against key exchange protocols has been widely recognized [2, 10, 39, 43].

Moeller and Langley [38] propose the use of a Signaling Cipher Suite Value (SCSV) in TLS that prevents version downgrade attacks when all versions provide transcript authentication. SSL 2.0 and 3.0 are being deprecated, partly to prevent version downgrades as these versions do not support TLS extensions [6, 41] and SSL 2.0 in any case does not provide transcript authentication. Similarly, ciphersuite hygiene is frequently discussed in standard documents [32, 34].

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 in the PKCS#1-encrypted pre-master secret. Klíma et al. [28] showed that many server implementations revealed 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.

Multi-ciphersuite security of SSH: Bergsma et al. [9] analyzed SSH in a multi-ciphersuite setting. They split the protocol into a negotiation phase NP and key-exchange phase SP, one for each value of $\pi.mode$. They show that if each combination $\mathtt{NP}\|\mathtt{SP}$ is ACCE secure, then $\mathtt{NP}\|\overrightarrow{\mathtt{SP}}$ is multi-ciphersuite ACCE secure. While they do not prove downgrade security per se, the result of [21] adapted to SSH corresponds to a proof of downgrade security for a *DP* predicate that guarantees that all negotiable ciphersuites and versions provide ACCE security.

Sharing of public keys is admissible under the condition that each sub-protocol is still secure in the presence of an auxiliary oracle with long-term key functionality, e.g., signing, to simulate all other sub-protocols. In our terminology, the protocols $\mathtt{NP}\|\mathtt{SP}$ of Bergsma et al. [9] are single-mode restrictions of $\mathtt{NP}\|\overrightarrow{\mathtt{SP}}$. After their extension with auxiliary oracles providing sufficient access to long-term key functionalities, they are 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. [9], our result allows to prove multi-ciphersuite ACCE security when not all sub-protocols in $\overrightarrow{\mathtt{SP}}$ are ACCE secure, as long as we restrict the protocol to configurations in *DP* that do not negotiate them (cf. Theorem 1).

Previous downgrade security theorems about TLS 1.2: Dowling and Stebila [21] model ciphersuite and version negotiation for TLS up to version 1.2 in the multi-ciphersuite setting introduced by Bergsma et al. [9]. In our model, their result corresponds to a proof of downgrade security for a *DP* 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, restated in §III-D, states that under such conditions multi-mode authentication implies downgrade security. This is a rather weak form of downgrade security, but as shown by Logjam, TLS 1.2 does not provide much stronger protection for clients. Servers that authenticate clients can however obtain stronger guarantees.

IX. 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. 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 [17] and Tamarin [37] 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 [16], 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 [7]. 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.

Acknowledgments

We thank Cas Cremers, Travis Cross, and Antoine Delignat-Lavaud for useful discussions, and the anonymous reviewers for their valuable comments. The TLS 1.3 downgrade countermeasures were formulated in collaboration with Eric Rescorla, Martin Thomson, and the IETF TLS working group. Karthikeyan Bhargavan is funded by the ERC grants CRYSP and CIRCUS. Christina Brzuska is grateful to NXP for supporting her chair for IT Security Analysis. This material is based in part upon work supported by the U.S. National Science Foundation under award EFRI-1441209; The Mozilla Foundation; and the Office of Naval Research under contract N00014-11-1-0470.

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*, 2015, pp. 5–17.

[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*, 2013, pp. 526–540.

[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*, 2013, pp. 305–320.

[5] 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.

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

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

[8] M. Bellare and P. Rogaway, "Entity authentication and key distribution," in *CRYPTO 1993*, 1993, pp. 232–249.

[9] 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*, 2014, pp. 369–381.

[10] 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*, 2015, pp. 535–552.

[11] 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.

[12] 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*, 2013, pp. 445–459.

[13] 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*, 2014, pp. 235–255.

[14] K. Bhargavan, C. Brzuska, C. Fournet, M. Green,

M. Kohlweiss, and S. Zanella-Bguelin, "Downgrade resilience in key-exchange protocols," Cryptology ePrint Archive, Report 2016/072, 2016, http://eprint.iacr.org/.

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

[16] Blanchet, "A computationally sound mechanized prover for security protocols," *Dependable and Secure Computing, IEEE Transactions on*, vol. 5, no. 4, pp. 193–207, 2008.

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

[18] R. Canetti and H. Krawczyk, "Security analysis of IKE's signature-based key-exchange protocol," in *CRYPTO 2002*, 2002, pp. 143–161.

[19] 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*, 2011, pp. 315–334.

[20] 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*, 2015, pp. 227–237.

[21] 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*, 2015, pp. 270–288.

[22] 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*, 2015, pp. 1197–1210.

[23] 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*, 2015, pp. 113–128.

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

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

[26] M. Just and S. Vaudenay, "Authenticated multi-party key agreement," in *ASIACRYPT 1996*, 1996, pp. 36–49.

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

[28] 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*. Springer, 2003, pp. 426–440.

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

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

[31] H. Krawczyk, K. G. Paterson, and H. Wee, "On the security of the TLS protocol: A systematic analysis," in *CRYPTO 2013*, 2013, pp. 429–448.

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

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

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

[35] 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*, 2012, pp. 62–72.

[36] C. Meadows, "Analysis of the internet key exchange protocol using the NRL protocol analyzer," in *1999 IEEE Symposium on Security and Privacy*, 1999, pp. 216–231.

[37] 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*, 2013, pp. 696–701.

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

[39] 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.

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

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

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

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

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

[45] ——, "The secure shell (SSH) transport layer protocol," IETF RFC 4253, 2006.

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