A preliminary version appears in CT-RSA 2010, Lecture Notes in Computer Science, Springer-Verlag, 2010.

Hash Function Combiners in TLS and SSL

Marc Fischlin Anja Lehmann Daniel Wagner

Abstract. The TLS and SSL protocols are widely used to ensure secure communication over an untrusted network. Therein, a client and server first engage in the so-called handshake protocol to establish shared keys that are subsequently used to encrypt and authenticate the data transfer. To ensure that the obtained keys are as secure as possible, TLS and SSL deploy *hash function combiners* for key derivation and the authentication step in the handshake protocol. A robust combiner for hash functions takes two candidate implementations and constructs a hash function which is secure as long as at least one of the candidates is secure. In this work, we analyze the security of the proposed TLS/SSL combiner constructions for pseudorandom functions resp. message authentication codes.

1 Introduction

Hash functions are an important primitive for cryptographic protocols and are currently used for various tasks that require, among others, collision resistance or, in keyed settings, behavior of a pseudorandom function or a MAC. However, recent attacks [WYY05, WY05, CR08, SSA+09] against the most widely deployed hash functions MD5 and SHA1 caused a decrease of confidence, especially concerning long-term security. Hence, approaches like robust combiners [Her05] which allow to obtain less vulnerable hash functions are of great interest and have triggered a series of research [BB06, Pie07, CRS+07, FL07, Pie08, FLP08].

In general, a hash combiner takes two hash functions H_0, H_1 and combines them into a failure-tolerant function such that this function remains secure as long as at least one of the two functions H_0 or H_1 is secure. For example, the classical combiner for collision-resistance simply concatenates the outputs of both hash functions $\mathsf{Comb}(M) = H_0(M)||H_1(M)$. If a hash function is supposed to be used as a pseudorandom function (PRF), then the exclusive-or of the outputs $\mathsf{Comb}(k_0||k_1, M) = H_0(k_0, M) \oplus H_1(k_1, M)$ with independent keys k_0, k_1 yields a robust design. Combiners that preserve even multiple properties in a robust manner where proposed in [FL08, FLP08].

Interestingly, the fact that combiners give better security assurances has been acknowledged by the designers of TLS and its predecessor SSL, long before they have been investigated more thoroughly by theoreticians. Both TLS and SSL use various combinations of MD5 and SHA1 instead of relying only on a single hash function. The specification of TLS even explicitly states: "In order to make the PRF as secure as possible, it uses two hash algorithms in a way which *should* guarantee its security if either algorithm remains secure" [TLS99] .

The SSL protocol [SSL94] was published in 1994 by Netscape to provide secure communication between two parties over an untrusted network, and subsequently formed the basis for the TLS protocol [TLS99, TLS06]. Nowadays, both protocols are ubiquitously present in various applications such as electronic banking, online shopping or secure data transfer. However, neither TLS nor SSL were accompanied with rigorous security proofs. An important step was recently done by Morrissey et al. [MSW08] and Gajek et al. [GMP+08] who gave the first security analysis of the handshake protocol of TLS. The handshake protocol is the essential part of TLS/SSL as it allows a client and server to negotiate security parameters, such as shared symmetric keys or trusted ciphers, without having any common secrets yet. The established keys and cryptographic algorithms are subsequently used to protect the data transfer, i.e., the confidentiality and authenticity of the entire communication relies on the security of the key agreement. Thus, it is of crucial importance that the handshake protocol provides reliable parameters. Ideally, this statement should be fortified by comprehensive security proofs.

Our Results. In this work, we scrutinize the design of the (non-standard) hash combiners, deployed in the TLS and SSL handshake protocols, regarding the suitability for the respective purposes. As already mentioned, secure key derivation is one of the main tasks of the handshake phase. Both TLS and SSL use hash combiners to compute the master secret out of the pre-master secret, which is assumed to be a shared random string. To achieve secure key-derivation, robustness with respect to pseudorandomness is required.

While TLS (mainly) reverts to the standard design for PRF combiners, i.e., it xors the outputs of the two hashes, SSL applies the cascade $H_0(k, (H_1(k, M))$ as the pseudorandom function for key derivation. For SSL we prove that the combiner is not robust and not even preserving, i.e., even two secure PRFs may yield an insecure combiner. This stems from the fact that both hash functions are invoked with the same master key. By using individual keys for each underlying function, we show that the security of the SSL combiner is somewhat between robustness and property-preservation. In the case of TLS, we prove that the combiner is a secure PRF if either of H_0, H_1 is a pseudorandom function. Interestingly, the TLS construction is neither optimal in terms of security nor efficiency. We therefore also discuss possible tweaks to obtain better security bounds while saving on computation.

TLS and SSL also use hash combiners for the finished message in the handshake protocol, which is basically a message authentication code generated for the shared master secret and all previous handshake messages. This concludes the key exchange phase in TLS/SSL and authenticates the previous communication. Ideally, the combiners used for this purpose should be robust for MACs, i.e., rely only on unforgeability instead of pseudorandomness of the hash function.[1] We show that in TLS the combiner for

[1]The devil's advocate may claim that we can already start from the assumption that one of the hash function is a PRF, as we require this for the key derivation step anyway. However, it is a common principle to revert to the minimal requirements for such sub protocols and their designated purpose. Suppose, for example, that both hash functions turn out to be *not* pseudorandom, that key derivation becomes insecure and confidentiality of the subsequently transmitted data is breached. Then, if one of

authentication requires the additional assumption of at least one hash function being collision resistant. The combiner used in SSL is again neither robust nor preserving, due to the same problem of using the master secret as key for both functions. We discuss that a modified version which splits the key into independent halves, is a secure MAC when at least one hash function is simultaneously unforgeable and collision resistant.

In summary, we give the first formal treatment of the hash combiners deployed in the TLS and SSL protocols. Our results essentially show that the choices in TLS are sound as they follow common design criteria for such combiners (but still leave space for improvements), whereas the SSL design for combiners requires much stronger assumptions. Our result, together with other steps like the security proofs in [MSW08, GMP+08], strengthen the confidence in the important protocols TLS and SSL.

2 Preliminaries

In this section we present the preliminaries for our investigation of the combiners in SSL/TLS.

2.1 Hash Functions and Their Properties

Since we give all results in terms of concrete security we adopt Rogaway's approach [Rog06] of defining hash functions as single instances (instead of families) and considering constructive reductions between security properties. For security notions without secret keys like collision-resistance the adversary is implicitly based on (the description of) H, whereas for security properties involving secret keys like pseudorandomness or the MAC property, the adversary also gets black-box access to the hash function $H(k,\cdot)$ with secret key k (we often write $H(k||\cdot)$ if the key is simply prepended to the message). In this case we call H a keyed hash function and usually denote the key space by K.

Most recent hash functions such as MD5, SHA1 apply the Merkle-Damgård construction [Mer90, Dam90] to obtain a variable-input length function out of a fixed-input length compression function $h : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^n$ and an initial vector IV. To compute a digest one divides (and possibly pads) the message $M = m_0 m_1 \dots m_{k-1}$ into blocks m_i of ℓ bits and computes the digest $H(M) = \mathrm{iv}_k$ as

$$\mathrm{iv}_0 = \mathrm{IV}, \qquad \mathrm{iv}_{i+1} = h(\mathrm{iv}_i, m_i) \qquad \text{for } i = 0, 1, \dots, k-1.$$

Collision-Resistance. Let $H : \{0,1\}^* \to \{0,1\}^n$ be a hash function. The *collision-finding advantage* of an adversary $\mathcal{A}$ is

$$\mathbf{Adv}^{\mathrm{cr}}_H(\mathcal{A}) := \mathrm{Prob}\left[(M, M') \leftarrow \mathcal{A}() : M \neq M' \wedge H(M) = H(M')\right].$$

We again note that, formally, for any hash function there is a very efficient algorithm $\mathcal{A}$ with advantage 1, namely, the one which has a collision hardwired into it and simply

the function is nonetheless still a good MAC, a secure authentication step in the finished message via the robust MAC-combiner would still guarantee authenticity of the designated partner.

outputs this collision. However, based on current knowledge it is usually infeasible to specify this algorithm constructively (cf. [Rog06]).

Pseudorandomness. Let $H : K \times \{0,1\}^* \to \{0,1\}^n$ be a keyed hash function with key space K. We define the advantage of a distinguisher $\mathcal{A}$ as

$$\mathbf{Adv}_H^{\mathrm{prf}}(\mathcal{A}) = \left|\mathrm{Prob}\left[\mathcal{A}^{H(k,\cdot)} = 1\right] - \mathrm{Prob}\left[\mathcal{A}^{f(\cdot)} = 1\right]\right|$$

where the probability in the first case is over $\mathcal{A}$'s coin tosses and the choice of $k \xleftarrow{\$} K$, and in the second case over $\mathcal{A}$'s coin tosses and the choice of the random function $f : \{0,1\}^* \to \{0,1\}^n$.

Message Authentication (Unforgeability). Let $H : K \times \{0,1\}^* \to \{0,1\}^n$ be a keyed (deterministic) hash function with key space K. We define the *forgeability advantage* of an adversary $\mathcal{A}$ as

$$\mathbf{Adv}_H^{\mathrm{mac}}(\mathcal{A}) = \mathrm{Prob}\left[k \xleftarrow{\$} K, (M,\sigma) \leftarrow \mathcal{A}^{H(k,\cdot)} : H(k,M) = \sigma \wedge M \text{ not queried}\right]$$

Hash Function Combiners. A hash function combiner Comb for hash functions H_0, H_1 "merges" the two functions H_0, H_1 into a single hash function. The combiner is called *preserving* [Her05] for some property like collision-resistance if Comb^{H_0,H_1} has this property given that *both* hash functions have this property. In a sense, this ensures a minimalistic security guarantee. The combiner is called *robust* [Her05] if it obeys the property if at least *one* of the two functions H_0, H_1 has the corresponding property. Note that, in terms of our concrete security statements, collision-resistance robustness for example is formulated by demanding that the probability of finding collisions in a combiner is bounded from above by the minimum of finding collisions for the individual hash functions.

2.2 HMAC

Each hash function can be used as a pseudorandom function or MAC by replacing the initial value IV with a randomly chosen key k of the same size. A more convenient technique was proposed by Bellare et al. [BCK96a] with the HMAC/NMAC algorithms, which are message authentication codes built from iterated hash functions. Recall that a MAC takes a secret key k, message m and outputs a tag σ. The HMAC algorithm takes, in its more general version, two keys $k_{\mathrm{in}}, k_{\mathrm{out}}$ and applies an iterated hash function H like $\mathsf{MD5}$ and $\mathsf{SHA1}$ in a nested manner:

$$\mathsf{HMAC}(k_{\mathrm{in}}, k_{\mathrm{out}})(M) = H(IV, k_{\mathrm{out}} || H(IV, k_{\mathrm{in}} || M)) \tag{1}$$

In practice, HMAC typically uses only a single key k from which it derives dependent keys $k_{\mathrm{in}} = k \oplus \mathsf{ipad}$ and $k_{\mathrm{out}} = k \oplus \mathsf{opad}$ for fixed constants $\mathsf{ipad} = 0x3636\ldots36, \mathsf{opad} = 0x5c5c\ldots5c$.

Originally, Bellare et al. [BCK96a] proved HMAC – resp. its theoretical counterpart NMAC – to be pseudorandom functions when the underlying compression function h is pseudorandom and collision-resistant. Subsequently, the proof was restated on the sole assumption that the compression function is pseudorandom [Bel06]. As the security claims are given for NMAC, Bellare [Bel06] introduced the notion of a "dual" pseudorandom function function $\bar{h} : \{0,1\}^n \times K \to \{0,1\}^n$ with $\bar{h}(m,k) = h(k,m)$. If both $\bar{h}$ and h are pseudorandom, the security of NMAC carries over to HMAC:

Lemma 2.1 *Let $h : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^n$ be a compression function with key space $\{0,1\}^n$. Let $IV \in \{0,1\}^n$ be a fixed initialization vector, and let $\mathsf{HMAC} : \{0,1\}^n \times \{0,1\}^* \to \{0,1\}^n$ be defined as in (1). For any adversary $\mathcal{A}$ against HMAC that makes q queries each of at most l blocks and runs in time at most t, there exist adversaries $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ such that*

$$\mathbf{Adv}^{prf}_{\mathsf{HMAC}}(\mathcal{A}) \;\leq\; 2\mathbf{Adv}^{prf}_{\bar{h}}(\mathcal{A}_1) + \mathbf{Adv}^{prf}_{h}(\mathcal{A}_2) + \binom{q}{2}\left[2l \cdot \mathbf{Adv}^{prf}_{h}(\mathcal{A}_3) + 2^{-n}\right]$$

where $\mathcal{A}_1$ makes a single query IV and runs in time at most t. $\mathcal{A}_2$ makes at most q queries and runs in time at most t, while $\mathcal{A}_3$ makes at most 2 oracle queries and runs in time at most $\mathcal{O}(lT_h)$ where T_h denotes the time required for one evaluation of h.

For the single-keyed HMAC-version, the security of $\bar{h}$ must hold for related-key attacks as well. That is, the adversary is granted access to two oracles $\bar{h}(k \oplus \mathsf{opad}, \cdot), \bar{h}(k \oplus \mathsf{ipad}, \cdot)$ with dependent keys.

2.3 The SSL/TLS Handshake Protocol

The SSL and TLS protocols consist of two layers: the record layer and the handshake protocol. The record layer encrypts all data with a cipher and session key that have been negotiated by the handshake protocol. Thus the handshake protocol is a key-exchange protocol layered above the record layer and initializes and synchronizes a cryptographic state between a server and a client. Both versions of the handshake protocol, for TLS and for SSL, vary mainly in the implementation of the exchanged messages, i.e., the overall structure of the handshake part is the same and can be summarized as the sequence of the following steps [Res01](see also Figure 1):

(1) The client conveys its willingness to engage in the protocol by sending a list of supported cipher algorithms and a random number, that is subsequently used for key-derivation.

(2) The server responds by choosing one of the proposed ciphers, and sending its certified public key as well as a random nonce.

(3) The client verifies the validity of the received certificate and sends a randomly chosen *pre-master secret* encrypted under the server's public key.

(An alternative to having the client choose the pre-master secret is to engage in a key exchange protocol like signed Diffie-Hellman. Since our analysis below only assumes that the pre-master secret is random we omit the details about its generation.)

(4) Both client and server individually compute a *master secret* from the exchanged random nonces and the pre-master secret. Once the master key is computed, it can be used to obtain further application keys.

(5+6) Finally, the master secret is confirmed by the *finished message*, where each party sends a MAC over the transcript of the conversation using the new master key. This is also the first transmission which uses the secure channel for the derived keys.

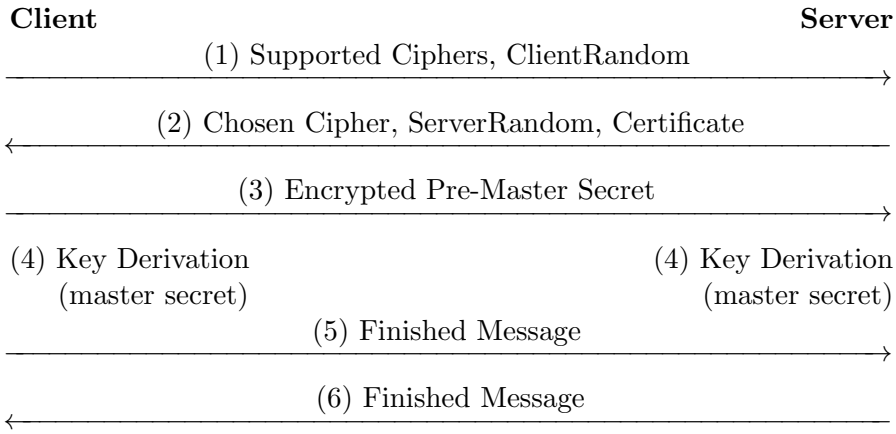

Figure 1: Overview of the TLS/SSL handshake protocol [Res01]

3 Derivation of the Master Secret

In this section we analyze the functions that are deployed by TLS and SSL to derive a secret master key from a shared pre-master key. The basic requirement of key derivation is that the obtained key should be indistinguishable from a randomly chosen one. In particular, the key-derivation function must be pseudorandom. For more discussion see [Kra08]. We will show that the combiner proposed by TLS is PRF-robust, i.e., security of one of the underlying hash function suffices, whereas the SSL combiner requires assumptions on both hash functions in order to produce random looking output.

3.1 The PRF-Combiner used in TLS

The TLS key derivation obtains the master secret (ms) from the pre-master secret (pms) by invoking the following hash combiner:

$$ms = \mathsf{Comb}^{\mathsf{MD5},\mathsf{SHA1}}_{\mathsf{TLS-prf}}(pms, \text{“master secret”}, \text{ClientRandom}||\text{ServerRandom})[0..47]$$

The pre-master secret is assumed to be a random value both parties have agreed upon, and *ClientRandom* and *ServerRandom* are public random nonces exchanged in the handshake protocol. By introducing a specific label (here “master secret”) to the input, the

combiner can subsequently be used for further (key-derivation) computations, while guaranteeing distinct inputs for each application. The appendix [0..47] indicates that the master secret consists of the first 48 bytes of the combiners output.

Basically, the combiner $\mathsf{Comb}^{H_0,H_1}_{\mathsf{TLS-prf}}$ xors the output of a function P which gets called twice based on two distinct hash functions H_0 and H_1. To this end, the combiner also splits the key $K = k_0||k_1$ with $|k_1| = |k_0|$ into independent halves:

$$\mathsf{Comb}^{H_0,H_1}_{\mathsf{TLS-prf}}(k_0||k_1, M) = \mathsf{P}_{H_0}(k_0, M) \oplus \mathsf{P}_{H_1}(k_1, M) \tag{2}$$

The underlying function P_{H_b} makes several queries to HMAC_{H_b} and produces byte strings of (arbitrary) length that is a positive multiple of n.

$$\mathsf{P}_{H_b}(k, M) = \mathsf{HMAC}_{H_b}(k, A(1)||M) \;||\; \mathsf{HMAC}_{H_b}(k, A(2)||M)|| \ldots \tag{3}$$

with $A(0) = M$ and $A(i) = \mathsf{HMAC}_{H_b}(k, (A(i-1))$.

Analysis of $\mathsf{Comb}^{H_0,H_1}_{\mathsf{TLS-prf}}$. We show that the TLS-combiner for key derivation is a pseudorandom function if at least one of the two hash functions H_0, H_1 is based on a pseudorandom compression function. To this end, we first show that P_{H_b} inherits the pseudorandomness of the underlying hash function.

Note that the P_{H_b} construction uses the HMAC transform to obtain a PRF, which gets keyed via the input data, out of a standard hash function H_b with fixed IV. It was shown in [Bel06] that HMAC is a pseudorandom function, when the underlying compression-function is a *dual PRF*, i.e., it has to be a secure PRF when keyed by either the data input or the chaining value. Thus, while functional-wise HMAC uses the cryptographic hash function only as a black-box, the security guarantee is still based on the underlying compression function h_b. We therefore consider each hash function $H_b : \{0,1\}^* \to \{0,1\}^n$ as the Merkle-Damgård iteration of a compression function $h_b : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^n$. By applying Lemma 2.1 we can conclude that HMAC_{H_b} is a pseudorandom function, when h_b is a dual PRF.

Next, we show that the design of the P_H construction preserves the pseudorandomness of HMAC_H. For a modular analysis – and for the sake of readability – we simplify the description of P_H by replacing HMAC and the hash function H by the same function H, and prove that the modified function P'_{H} is a pseudorandom function if H is. Furthermore, we make a rather syntactical change of P to obtain a function that is efficiently computable on its own: According to the TLS specification, the P construction produces output of arbitrary length from which the combiner takes as much bytes as required, e.g., the first 48 bytes in case of the derivation of the master secret. In the following we slightly deviate from that notation and assume that P gets also parametrized by an integer c which indicates that an output of length $c \cdot n$ is requested. Overall, we analyze the following function P':

$$\begin{aligned} &\mathsf{P}'_{\mathsf{H}}(k, M, c) = \\ &\quad \mathsf{H}(k, A(0) \;||\; M) \;\; || \;\; \mathsf{H}(k, A(1) \;||\; M) \;\; || \;\; \ldots \;\; || \;\; \mathsf{H}(k, A(c-1) \;||\; M) \end{aligned} \tag{4}$$

where $A(0) = M, \quad A(i) = \mathsf{H}(k, (A(i-1))$.

Lemma 3.1 *Let* $\mathsf{H} : \{0,1\}^n \times \{0,1\}^* \to \{0,1\}^n$ *be a pseudorandom function with key space* $\{0,1\}^n$*, and let* $\mathsf{P}'_{\mathsf{H}} : \{0,1\}^n \times \{0,1\}^* \to \{0,1\}^{c \cdot n}$ *be defined by (4) above. For all adversaries* $\mathcal{A}$ *running in time* t*, making* q *queries of length at most* l *and with* $c \leq c_{\max}$*, there exist an adversary* $\mathcal{B}$ *such that*

$$\mathbf{Adv}^{prf}_{\mathsf{P}'}(\mathcal{A}) \;\leq\; \mathbf{Adv}^{prf}_{\mathsf{H}}(\mathcal{B}) + q \cdot \binom{c_{max}}{2} \cdot 2^{-n}$$

where $\mathcal{B}$ *makes at most* $2c_{max} \cdot q$ *queries, each of length at most* $l + n$ *and runs in time at most* $t + \mathcal{O}(c_{max})$*.*

Proof. Assume that there is an adversary $\mathcal{A}$ that can distinguish the function $\mathsf{P}'_{\mathsf{H}}(k,\cdot)$ from a random function $F : \{0,1\}^* \to \{0,1\}^n$ with advantage $\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{P}'}(\mathcal{A})$. Given $\mathcal{A}$ we show how to obtain an adversary $\mathcal{B}$ against the underlying hash function $\mathsf{H}(k,\cdot)$. Recall that $\mathcal{A}$ has black-box access to an oracle that is either the keyed construction $\mathsf{P}'_{\mathsf{H}}(k,\cdot,\cdot)$ or a random function $F : \{0,1\}^* \to \{0,1\}^*$ (where, formally, F also takes the parameter c as additional input and outputs strings of length cn). The distinguisher $\mathcal{B}$ has to simulate this oracle with the help of its own oracle, which is either the keyed hash-function $\mathsf{H}(k,\cdot)$ or a random function $f : \{0,1\}^* \to \{0,1\}^n$. To this end, for any query (M,c) of $\mathcal{A}$, the adversary $\mathcal{B}$ mimics the construction P' but replaces each evaluation of the underlying hash function H by the response of its oracle on the corresponding query. If $\mathcal{A}$ stops outputting its guess d, algorithm $\mathcal{B}$ stops with output d too.

If the oracle of $\mathcal{B}$ was the hash function H, then $\mathcal{B}$ perfectly simulates the construction P'. Thus, the output distribution of $\mathcal{B}$ equals the one of $\mathcal{A}$ with access to P':

$$\mathrm{Prob}\Big[\,\mathcal{B}^{\mathsf{H}(k,\cdot)} = 1\Big] = \mathrm{Prob}\Big[\,\mathcal{A}^{\mathsf{P}'_{\mathsf{H}}(k,\cdot,\cdot)} = 1\Big]\,.$$

In the case that the oracle of $\mathcal{B}$ was the truly random function f, we have to show that processing its random answers in the P' construction yields random values again. Recall that for each query (M,c) the adversary $\mathcal{B}$ now computes the sequence $f(A(0)||M)\;||\;f(A(1)||M)\;||\ldots||\;f((A(c-1)||M)$ where $A(i) = f(A(i-1))$ starting with $A(0) = M$. As long as $A(i) \neq A(j)$ for all $i \neq j \in \{0,1,\ldots c-1\}$ holds for each query, the function f gets evaluated in the outer iterations on distinct and unique values, such that the corresponding outputs from f are independently and uniformly distributed. Thus, it remains to show that the probability for collisions on the $A(i)$ values, which are derived using f in a cascade, is small. Assume that for a query (M,c) a collision occurred, i.e., there exist (unique) indices $i^* \in \{0,\ldots,c-1\}$ and $j^* \in \{0,\ldots,i^*-1\}$ such that $f(A(i^*-1)) = A(j^*)$ but $A(i^*-1) \neq A(j)$ for all $j = 0,1,\ldots,i^*-2$. That is, f has never been invoked on the value $A(i^*-1)$ but maps to an value $A(j^*)$ which is an previous answer of (the cascade of) f. Since f is a truly random function, such a collision can only occur with probability $q \cdot \binom{c_{\max}}{2} \cdot 2^{-n}$ where q denotes the number of $\mathcal{A}$ queries and $c_{\max}$ is the largest value for c that appeared in the simulation. Overall, the output distribution of $\mathcal{B}^f$ results from

$$\begin{aligned}\mathrm{Prob}\Big[\,\mathcal{B}^f = 1\Big] \;&\leq\; \mathrm{Prob}\Big[\,\mathcal{B}^f = 1 \;\Big|\; \text{no Collision}\Big] + \mathrm{Prob}\,[\,\text{Collision}] \\ &=\; \mathrm{Prob}\,[\,\mathcal{A}^F = 1] + q \cdot \binom{c_{\max}}{2} \cdot 2^{-n}.\end{aligned}$$

Thus, $\mathcal{B}$ distinguishes H from f with probability:

$$\begin{aligned}&\mathrm{Prob}\Big[\mathcal{B}^{\mathsf{H}(k,\cdot)}=1\Big]-\mathrm{Prob}\Big[\mathcal{B}^{f}=1\Big]\\&\quad\geq \mathrm{Prob}\Big[\mathcal{A}^{\mathsf{P}_{\mathsf{H}}(k,\cdot,\cdot)}=1\Big]-\mathrm{Prob}\big[\mathcal{A}^{F}=1\big]-q\cdot\binom{c_{\max}}{2}\cdot 2^{-n}.\end{aligned}$$

This proves the claim. □

Putting Lemma 2.1 and Lemma 3.1 together, we now obtain that the pseudorandomness of h_b is preserved by the corresponding construction HMAC_{H_b} and, in turn, by $\mathsf{P}'_{\mathsf{HMAC}_{H_b}}$ which equals P_{H_b}. Furthermore, XOR is a robust combiner for pseudorandom functions, and thus, if least one of $\mathsf{P}_{H_0}, \mathsf{P}_{H_1}$ is a PRF, also $\mathsf{Comb}^{H_0,H_1}_{\mathsf{TLS-prf}}$ provides outputs that are indistinguishable from random. This, together with the fact that the key is divided into independent halves, implies the following theorem:

Theorem 3.2 *Let $H_b : \{0,1\}^n \times \{0,1\}^* \to \{0,1\}^n$ for $b \in \{0,1\}$ be a hash function with underlying compression function $h_b : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^n$. Let $\mathsf{Comb}^{H_0,H_1}_{\mathsf{TLS-prf}}$ be defined as in (2). For all adversaries $\mathcal{A}$ running in time t, making q queries of length at most l and such that $c \leq c_{\max}$, there exist adversaries $\mathcal{A}_0, \mathcal{A}_1$ such that*

$$\begin{aligned}&\mathbf{Adv}^{prf}_{\mathsf{Comb}_{\mathsf{TLS-prf}}}(\mathcal{A})\\&\quad\leq\quad \min\Big\{\mathbf{Adv}^{prf}_{\mathsf{HMAC}_{h_0}}(\mathcal{A}_0), \mathbf{Adv}^{prf}_{\mathsf{HMAC}_{h_1}}(\mathcal{A}_1)\Big\}+q\cdot\binom{c_{max}}{2}\cdot 2^{-n}\end{aligned}$$

where each of $\mathcal{A}_0, \mathcal{A}_1$ makes at most $2c_{max}\cdot q$ queries of length at most $l+n$ and runs in time at most $t+\mathcal{O}(c_{max}(1+2q\cdot T_{\bar{b}}))$ where $T_{\bar{b}}$ denotes the time required for one evaluation of $\mathsf{P}_{\bar{b}}$ (as defined in (3)).

Improvements. When the combiner $\mathsf{Comb}^{H_0,H_1}_{\mathsf{TLS-prf}}$ is used for key derivation, the underlying construction P ensures that sufficiently many output bytes are produced. However for the purpose of range extension of a PRF, the construction P is neither optimal in terms of efficiency nor security. Namely, if one assumes HMAC_H to be a secure PRF, one could simply augment the input M by a fixed-length encoded counter $\langle i\rangle$, which ensures distinct inputs for each PRF evaluation:

$$\mathsf{P}^*_{H_b}(k,M) = \mathsf{HMAC}_{H_b}(k, M||\,\langle 1\rangle)\;||\;\mathsf{HMAC}_{H_b}(k, M||\,\langle 2\rangle)||\ldots$$

Replacing P with P^* would result in better security bounds, as one gets rid of the probability $q\cdot\binom{c_{\max}}{2}\cdot 2^{-n}$ of a collision on the $A(i)$ values. In terms of efficiency, the above construction only requires half of the PRF evaluations as needed in the original P function.

Another solution is to use solely the outputs of $A(\cdot)$, i.e., without feeding them into HMAC again:

$$\mathsf{P}^*_{H_b}(k,M) = A(1)\;||\;A(2)\;||\;A(3)\;||\;\ldots$$

with $A(i)$ being the i-th cascade of $\mathsf{HMAC}(k, M)$ as defined in (3). With this construction one inherits the same security bound as in the original solution, but invokes HMAC after the first evaluation only one shorter inputs, e.g., 128 bits in the case of MD5 and 160 bits for SHA1, which decreases the computational costs.

3.2 The PRF-Combiner used in SSL

In the SSL protocol the following construction gets repeated until sufficient key material for the master secret is generated:

$$\begin{aligned} ms \;=\; & \mathsf{MD5}(pms||(\mathsf{SHA1}(\text{“A”}||pms||\text{ClientRandom}||\text{ServerRandom}))|| \\ & \mathsf{MD5}(pms||(\mathsf{SHA1}(\text{“BB”}||pms||\text{ClientRandom}||\text{ServerRandom}))|| \\ & \mathsf{MD5}(pms||(\mathsf{SHA1}(\text{“CCC”}||pms||\text{ClientRandom}||\text{ServerRandom}))||\cdots \end{aligned}$$

Both functions get keyed by the input data, where in the case of the outer hash function the key is prepended to the message, and for the inner hash the key is somewhat embedded in the message. Due to length-extension attacks, key-prepending approaches must be accompanied by prefix-free encoding, otherwise the hash function can not serve as a pseudorandom function, as shown in [BCK96b]. For the analysis we assume that the hash function takes care of that issue, and thus that a hash function $H_b : \{0,1\}^* \to \{0,1\}^n$ is a secure PRF when keyed via the first n bits of the data input.

On a more abstract level, each repetition of the SSL-combiner above for prefixes “A”, “BB”, “CCC” etc. can be represented as the following construction:

$$\mathsf{Comb}^{H_0,H_1}_{\mathsf{SSL-prf}}(k, M) = H_0(k \;||\; H_1(k||M)), \tag{5}$$

e.g., where $H_1(k||M)$ implements $\mathsf{SHA1}(\text{“CCC”}||k||M)$ for the fixed value “CCC”. To be a robust combiner for pseudorandom functions, the SSL-combiner needs to be robust for H_0 and each such function H_1. From now on we fix an arbitrary H_1.

Analysis of $\mathsf{Comb}^{H_0,H_1}_{\mathsf{SSL-prf}}$. The cascade $\mathsf{Comb}^{H_0,H_1}_{\mathsf{SSL-prf}}$ of two hash functions is not a robust design for pseudorandomness, because as soon as the outer function becomes insecure the combiner, too, can be easily distinguished from a random function: Consider as an example the constant function $H_0(x) = 0^n$ that maps any input to zeros, which is obviously distinguishable from random. Then, also the combiner $\mathsf{Comb}^{H_0,H_1}_{\mathsf{SSL-prf}}(k, M) = H_0(k||H_1(k||M))$ becomes a constant function, independently of the strength of the inner hash function H_1. Hence,

Proposition 3.3 *The combiner* $\mathsf{Comb}^{H_0,H_1}_{\mathsf{SSL-prf}}$ *is* not *PRF-robust.*

Actually, $\mathsf{Comb}^{H_0,H_1}_{\mathsf{SSL-prf}}$ is not even PRF-preserving, i.e., there exist two functions H_0, H_1 that are both secure pseudorandom functions, but become easily distinguishable when used in the SSL-combiner. The problem arises from the fact that the same secret

key is used for both functions, which contradicts the general design paradigm of provably robust combiners.

For the counter example let $H_1 : K \times \{0,1\}^* \to \{0,1\}^n$ be a pseudorandom function. Define $H_0(k,x)$ now as follows: if $x = H_1(k, 0^n)$ then return 0^n, else output $H_1(k||1||x)$. Then H_0 basically inherits the pseudorandomness of H_1 because any distinguisher with access to $H_0(k,\cdot)$ only retrieves replies $H_1(k||1||x)$ to queries $x \in \{0,1\}^*$, unless it is able to predict the value $H_1(k||0^n)$. The latter would contradict the pseudorandomness of H_1, though. But when both functions are combined into $H_0(k \;||\; H_1(k||M))$, the combiner returns 0^n for input 0^n and is obviously therefore not a pseudorandom function.

In order to allow any reasonable statement about the security of the construction $\mathsf{Comb}^{H_0,H_1}_{\mathsf{SSL-prf}}$, we assume in the following that the combiner splits the key into two independent halves, and invokes the hash functions on distinct shares:

$$\mathsf{Comb}^{H_0,H_1}_{\mathsf{SSL-prf}^*}(k_0||k_1, M) = H_0(k_0 \;||\; H_1(k_1||M))$$

Note that the first discussed counter example is still valid, as it did not require any dependencies of the individual keys. Thus, even $\mathsf{Comb}^{H_0,H_1}_{\mathsf{SSL-prf}^*}$ is not a robust combiner in general. However, the security can be considered to be somewhat above property-preservation, since we can relax the assumption on one hash function while the combiner still preserves the pseudorandomness property of the stronger function:

PRF + weakCR = PRF. In the case that the outer hash function H_0 is a secure pseudorandom function, the inner hash function only needs to ensure that for distinct queries $M \neq M'$ of an adversary to the combiner, the function H_0 gets evaluated on different values too, i.e., $H_1(k,M) \neq H_1(k,M')$ holds for $M \neq M'$. Thus, it suffices for H_1 to be *weakly collision-resistant*, which is defined similarly to collision-resistance, except that here the function is keyed with a secret key and the adversary only gets black-box access to the function. Even though weakCR is a weaker assumption than standard CR, and it is for fixed input-length functions implied by the MAC security, it might suffer for variable length-inputs from the strong attacks against CR [Hir04]. In particular, MD5 and SHA1 are still assumed to be good pseudorandom functions but lack security against weakCR attacks, which was also the reason to restate the proof of HMAC in [Bel06].

weakPRF +PRF = PRF. If the inner hash function H_1 is a pseudorandom function, an adversary that queries the combiner gets to see images of H_0 only for random domain points. Thus, it is not necessary that the outer function is a full-fledged PRF as well. In this case, already the assumption of H_0 being a *weak pseudorandom function* is sufficient. This notion weakens the regular concept of PRFs in the sense that the adversary is only allowed to query the function an random inputs instead of values of his choice. Note that a weakPRF is significantly weaker than a PRF, as e.g., they can exhibit weak input points or be commutative.

insecure+insecure = PRF? One might ask if one can even start with two functions that both are not full PRFs itself, but add up to a secure PRF when used in the

combiner construction. It turns out that the SSL combiner allows for two almost entirely insecure hash functions to yield a secure PRF. However, this only holds for very artificial and tailored hash functions, hence, the impact on the security statements for practical considerations is quite limited. We also stress that this is not a particular benefit of the SSL design, as we can obtain similar examples for the TLS combiner as well. We discuss both examples in the Appendix A.

3.3 Application Key Derivation in TLS and SSL

Both combiners $\mathsf{Comb}^{H_0,H_1}_{\mathsf{TLS-prf}}$ and $\mathsf{Comb}^{H_0,H_1}_{\mathsf{SSL-prf}}$ are used to obtain a shared master secret from a pre-shared key. However, subsequently, the same functions are deployed to derive further keys, e.g., for encryption or message authentication. To this end, the freshly computed master secret is used instead of the pre-master secret that was assumed to be a random value. For TLS we have shown that the combiner $\mathsf{Comb}^{H_0,H_1}_{\mathsf{TLS-prf}}$ provides a master secret that is indistinguishable from random when at least one hash function is a PRF. Thus, our result carries over to the application key derivation, that uses the combiner with the derived master secret. The same holds for SSL, but under stronger assumptions on the underlying hash functions.

4 Finished-Message

In this section we investigate the TLS/SSL combiners that are used to compute the so-called *finished*-message of the handshake protocols. The finished message is the last part of the key exchange and is realized by a message authentication code which is computed over the transcript of the previous communication. Thus, the combiners that are used for this application should optimally be robust for MAC, i.e., only rely on the unforgeability property instead of the stronger PRF-assumption.

We note that the finished message itself is already secured through the negotiated application keys. This complicates the holistic security analysis of this step. But since we are at foremost interested in the design of the combiners and their designated purpose, we only touch this issue briefly at the end of Section 4.1 (where we address this issue for TLS; the same discussion holds for SSL).

4.1 The MAC-Combiner used in TLS

To compute the finished MAC, the TLS protocol applies the same combiner as for the derivation of the master secret, but already uses the new master key. As the key is known only at the very end of the protocol, the MAC cannot be computed iteratively during the communication. To circumvent the need of storing the entire transcript until the master secret is available, TLS hashes the transcript iteratively and then computes the MAC over the short hash value only:

$$\sigma_{\text{finished}} = \mathsf{Comb}^{\mathsf{MD5},\mathsf{SHA1}}_{\mathsf{TLS-prf}}\big(ms,\ \text{FinishedLabel},\ \mathsf{MD5}(\mathsf{transcript})||\mathsf{SHA1}(\mathsf{transcript})\big)[0..11]$$

A further input to the combiner is the FinishedLabel which is either the ASCII string "client" or "server", which ensures that the MAC values of both parties are different, otherwise an adversary could simply return a finished tag back to its sender. The appendix [0..11] indicates again that the first 12 bytes of the combiner output are used as the MAC.

Recall that the combiner $\mathsf{Comb}^{H_0,H_1}_{\mathsf{TLS-prf}}$ is based on the construction P which produces arbitrary length output by invoking the underlying hash function in an iterative and nested manner. However, this range extension is only necessary when the combiner is used for key derivation. To compute the finished message, only the first 12 byte of the combiners output are used, which is shorter than the digests of both applied hash functions (16 bytes for $\mathsf{MD5}$ and 20 bytes for $\mathsf{SHA1}$). Thus, we can omit the P part from the construction and simplify the combiner as follows:

$$\begin{aligned}&\mathsf{Comb}^{H_0,H_1}_{\mathsf{TLS-mac}}(k_0||k_1, M) = \\ &\quad \mathsf{HMAC}_{H_0}(k_0, H_0(M)||H_1(M)) \ \oplus \ \mathsf{HMAC}_{H_1}(k_1, H_0(M)||H_1(M))\end{aligned} \tag{6}$$

Verification for the above MAC-combiner is done by recomputing the tag and comparing it to the given tag.

Analysis of $\mathsf{Comb}^{H_0,H_1}_{\mathsf{TLS-mac}}$. We have already shown that the combiner construction $\mathsf{Comb}^{H_0,H_1}_{\mathsf{TLS-prf}}$, which can be seen as the more complex version of $\mathsf{Comb}^{H_0,H_1}_{\mathsf{TLS-mac}}$, is robust for pseudorandom functions. Thus, if one is willing to assume that at least one hash function behaves like a random function, the combiner can be used directly as a MAC, as well.

However, ideally, the combiner $\mathsf{Comb}^{H_0,H_1}_{\mathsf{TLS-mac}}$ should be a secure MAC on the sole assumption that at least one of the underlying hash functions H_0, H_1 is unforgeable rather than being a pseudorandom function. Unfortunately, hashing the transcript before the MAC gets computed, imposes another assumption on the hash functions (even when starting from the PRF assumption), namely at least one hash function needs to be collision-resistant. Otherwise an adversary could try to induce a collision on the input to the HMAC functions, which immediately gives a valid forgery for the entire MAC function. Under the assumption that such a collision is unlikely, we show that the combiner $\mathsf{Comb}^{H_0,H_1}_{\mathsf{TLS-mac}}$ is MAC-robust.

To this end, we first prove that the xor of two deterministic MACs (like HMAC_{H_b}) invoked directly with the message yields a robust combiner:

$$\mathsf{Comb}^{\mathsf{H}_0,\mathsf{H}_1}_{\oplus}(k_0||k_1, M) = \mathsf{H}_0(k_0, M) \oplus \mathsf{H}_1(k_1, M) \tag{7}$$

In the context of aggregate authentication, Katz and Lindell [KL08] gave a similar result by showing that multiple MAC tags, computed by (possibly) different senders on multiple (possibly different) messages, can be securely aggregated into a shorter tag by simply xoring them.

Lemma 4.1 *Let $\mathsf{H}_0, \mathsf{H}_1 : \{0,1\}^n \times \{0,1\}^* \to \{0,1\}^n$ be deterministic message authenticated codes, and let $\mathsf{Comb}^{\mathsf{H}_0,\mathsf{H}_1}_{\oplus}$ be defined by (7). For any adversary $\mathcal{A}$ against $\mathsf{Comb}^{\mathsf{H}_0,\mathsf{H}_1}_{\oplus}$*

making at most q queries and running in time at most t, there exist adversaries $\mathcal{A}_0, \mathcal{A}_1$ such that

$$\mathbf{Adv}^{mac}_{\mathsf{Comb}_\oplus}(\mathcal{A}) \leq \min\left\{\mathbf{Adv}^{mac}_{\mathsf{H}_0}(\mathcal{A}_0), \mathbf{Adv}^{mac}_{\mathsf{H}_1}(\mathcal{A}_1)\right\}$$

where $\mathcal{A}_b$ for $b = 0, 1$ makes at most q queries and runs in time at most $t + \mathcal{O}(qT_{\bar{b}})$ where $T_{\bar{b}}$ denotes the time for one evaluation of $\mathsf{H}_{\bar{b}}$.

Proof. We show that any adversary $\mathcal{A}$ against the combiner implies adversaries $\mathcal{A}_0, \mathcal{A}_1$ against both underlying MACs. Assume towards contradiction that an adversary $\mathcal{A}_{\mathsf{Comb}}$ after making q queries $M_1, \ldots, M_q$ to the $\mathsf{Comb}^{\mathsf{H}_0,\mathsf{H}_1}_\oplus(K, \cdot)$ oracle, outputs with some probability a tuple (M^*, σ^*) such that $\sigma^* = \mathsf{Comb}^{\mathsf{H}_0,\mathsf{H}_1}_\oplus(K, M^*)$ but M^* was never submitted to the combiner oracle. Given $\mathcal{A}_{\mathsf{Comb}}$ we construct a MAC adversary $\mathcal{A}_b$ against the MAC H_b for $b \in \{0, 1\}$. This adversary has oracle access to the function $\mathsf{H}_b(k_b, \cdot)$ and uses $\mathcal{A}_{\mathsf{Comb}}$ in a black-box way to produce its forgery. To this end, $\mathcal{A}_b$ first chooses a random key $k_{\bar{b}}$ for $\mathsf{H}_{\bar{b}}$ and then answers each query to the combiner with the help of its oracle access and the knowledge of $k_{\bar{b}}$. When $\mathcal{A}_{\mathsf{Comb}}$ holds, outputting a pair (M^*, σ^*), the adversary $\mathcal{A}_b$ computes its forgery (M^*, σ^*_b) with $\sigma^*_b = \sigma^* \oplus \mathsf{H}_{\bar{b}}(k_{\bar{b}}, M^*)$.

As M^* was not previously queried by $\mathcal{A}_{\mathsf{Comb}}$, the same holds for $\mathcal{A}_b$. Furthermore, as both MACs are deterministic, the value $\sigma^* = \mathsf{H}_0(k_0, M^*) \oplus \mathsf{H}_1(k_1, M^*)$ fixes two well-defined tags for $\mathsf{H}_0, \mathsf{H}_1$. Thus, $\mathcal{A}_b$'s output is equal to the value $\mathsf{H}_b(k_b, M^*)$ for the unknown key k_b and thereby constitutes a valid forgery. Since the adversary $\mathcal{A}$ yields forgers $\mathcal{A}_0, \mathcal{A}_1$ for both MACs, it follows that the advantage cannot exceed the advantage of the smaller of the two security bounds for the MACs. □

Complementing the above Lemma 4.1 with the probability of finding collisions on the concatenated combiner $H_0(M)||H_1(M)$ yields Theorem 4.2.

Theorem 4.2 *Let $H_0, H_1 : \{0,1\}^n \times \{0,1\}^* \to \{0,1\}^n$ be hash functions, and let $\mathsf{Comb}^{\mathsf{H}_0,\mathsf{H}_1}_{\mathsf{TLS-mac}}$ be defined by (6). For any adversary $\mathcal{A}$ against $\mathsf{Comb}^{\mathsf{H}_0,\mathsf{H}_1}_{\mathsf{TLS-mac}}$ making at most q queries and running in time at most t, there exist adversaries $\mathcal{A}_0, \mathcal{A}_1, \mathcal{B}_0, \mathcal{B}_1$ such that*

$$\begin{aligned}\mathbf{Adv}^{mac}_{\mathsf{Comb}_{\mathsf{TLS-mac}}}(\mathcal{A}) \leq \min&\left\{\mathbf{Adv}^{mac}_{\mathsf{HMAC}_{H_0}}(\mathcal{A}_0), \mathbf{Adv}^{mac}_{\mathsf{HMAC}_{H_1}}(\mathcal{A}_1)\right\} \\ +\quad &\min\left\{\mathbf{Adv}^{cr}_{H_0}(\mathcal{B}_0), \mathbf{Adv}^{cr}_{H_1}(\mathcal{B}_1)\right\}\end{aligned}$$

where $\mathcal{A}_b$ for $b = 0, 1$ makes at most q queries and runs in time at most $t + \mathcal{O}(qT_{\bar{b}})$ where $T_{\bar{b}}$ denotes the time for one evaluation of $\mathsf{HMAC}_{H_{\bar{b}}}$, and $\mathcal{B}_b$ runs in time $t + \mathcal{O}(qT_b)$.

Note that for both properties, unforgeability and collision-resistance, it suffices that either one of the hash functions has this property (instead of one hash function with obeying both property simultaneously). This is similar to the difference between weak and strong combiners in [FL08].

So far, we have reduced the security of the combiner $\mathsf{Comb}^{\mathsf{H}_0,\mathsf{H}_1}_{\mathsf{TLS-mac}}$ of H_0, H_1 to the collision-resistance of the hash functions and the unforgeability of the HMAC transforms HMAC_{H_0} and HMAC_{H_1}. Preferably, the security of HMAC_{H_b} should in turn only rely

on the unforgeability of the underlying hash resp. compression function. However, such a reduction for the plain HMAC transform is still unknown. The previous results for this issue either require stronger assumptions than MAC (yet, weaker than PRF), or additional keying-techniques for the compression function. In the following, we briefly recall the two most relevant approaches for our scenario.

An and Bellare [AB99] observed that HMAC can be used to build a VIL-MAC from a FIL-MAC (i.e., from an unforgeable compression function) when the secret key enters each compression function evaluation. As this result was shown for compression functions in the dedicated-key setting, one needs to transform compression functions without a dedicated-key input into keyed ones. This can be done as follows: Let $h : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^n$ be an unkeyed compression function. Then the keyed analogue $kh : \{0,1\}^n \times \{0,1\}^{n+\ell'} \to \{0,1\}^n$ is defined as $kh(k, y, x) = h(y, k||x)$ where $k \in \{0,1\}^n$ is the secret key, $y \in \{0,1\}^n$ the chaining value and $x \in \{0,1\}^{\ell'}$ with $\ell' = \ell - n$ the (shortened) message block[2]. This approach reduces the number of bits that can be processed in each iteration but allows to use the SHA1 and MD5 compression functions, which do not possess a dedicated key-input. Starting from such a keyed FIL-MAC, the HMAC variant that prepends k_{in} (resp. k_{out} at the final evaluation) to each message block, is proven to be a secure VIL-MAC [AB99]. When HMAC is used only with a single-key k, unforgeability of kh must hold against related key attacks for $kh(k \oplus \Delta_{\mathsf{opad}}, \cdot)$ and $kh(k \oplus \Delta_{\mathsf{ipad}}, \cdot)$. Overall, the first approach relies solely on the unforgeability assumption, but comes with a reduced throughput, e.g., when using SHA1, the HMAC variant that is keyed in each iteration, would require ≈ 1.5 times the compression function evaluations of the standard $\mathsf{HMAC}_{\mathsf{SHA1}}$.

The second approach does not require any modification of HMAC, i.e., it has the same efficiency, but makes stronger assumptions on the compression function: In [Bel06] Bellare proved that NMAC is a secure MAC if the underlying compression function h is a *privacy-preserving MAC* (PP-MAC) and the iteration of h is *computationally almost universal* (cAU). The former resembles the indistinguishability notion for encryption and requires that an adversary given a tag $\mathsf{Mac}(k, M_b)$ for chosen M_0, M_1 cannot determine b. It was shown that PP-MAC is a weaker assumption than PRF and cAU is a milder assumption than weakCR. Fischlin [Fis08] has shown that, alternatively, non-malleability and unpredictability of the compression function suffices, too. In both cases, however, in order to lift the security of NMAC to the single-key version of HMAC, one additionally needs that the dual compression function $\bar{h}$ used to derive $k_{\text{in}}, k_{\text{out}}$ somehow preserves these conditions.

The Problem of Chopping. Theorem 4.2 states that the TLS-combiner for the finished message is robust for message authentication codes even when starting from the unforgeability assumption which is significantly weaker than assuming a PRF. However, according to the TLS specification, not the entire output of the combiner is used as tag,

[2]Actually, An and Bellare proposed a transformation that keys the compression function via the chaining value, which would not allow black-box usage of the underlying hash function. We therefor swap the roles of chaining and key value, and assume that kh is a secure MAC when the key occupies the first n bits of each input.

but only the first 12 bytes. Since the unforgeability notion is not closed under chopping transformations, a shortened output of a MAC loses any security guarantees. To allow usage of a chopped fraction of the combiners output, on either has to assume that one of the underlying MACs is secure for truncation, or one needs to make the stronger assumption that at least one of the two hash functions is a secure PRF.

Is Unforgeability Enough? When using MACs in a stand-alone fashion, unforgeability clearly gives sufficient security guarantees. However, in TLS (and SSL) the tag for the finished message is computed under the master secret, from which further application keys for encryption and authentication are derived. The tag itself is now encrypted and authenticated with these derived keys. On one hand, this may help to prevent the tag in the finished message from leaking some information about the master secret. On the other hand, this causes critical circular dependencies between these values, possibly even enabling leakage of entire keys. This problem has already been noticed in other works (e.g., in [MSW08] where the analysis of the handshake protocol assumes that the tag is sent without securing it with the application keys; or more explicitly in the context of delayed-key authentication in [FL10]). It is beyond the scope of this work about combiners, though.

4.2 The MAC-Combiner used in SSL

The SSL-construction for the finished message resembles the HMAC construction, but appends the inner key to the message instead of prepending it. This stems from the same problem as in TLS, namely that the MAC should be computed iteratively as soon as the communication starts, although the necessary key is negotiated only at the end. To obtain a robust design, SSL uses the concatenation of the HMAC-like construction based on the $\mathsf{MD5}$ and $\mathsf{SHA1}$ functions:

$$\sigma_{\text{finished}} = \mathsf{HMAC}^*_{\mathsf{MD5}}(ms, \text{Label}||\mathsf{transcript}) \;||\; \mathsf{HMAC}^*_{\mathsf{SHA1}}(ms, \text{Label}||\mathsf{transcript})$$

where HMAC^*_H is defined as:

$$\mathsf{HMAC}^*_H(k, M) = H(k||\mathsf{opad}||\;\; H(M||k||\mathsf{ipad})) \tag{8}$$

with $\mathsf{opad}, \mathsf{ipad}$ being the same fixed patterns as in HMAC. The structure of HMAC^* then allows to accomplish the bulk of the computation without knowing the key k.

Overall, the MAC combiner of SSL can be described as follows:

$$\mathsf{Comb}^{H_0,H_1}_{\mathsf{SSL-mac}}(k, M) = \mathsf{HMAC}^*_{H_0}(k, M) \;||\; \mathsf{HMAC}^*_{H_1}(k, M) \tag{9}$$

Analysis of $\mathsf{Comb}^{H_0,H_1}_{\mathsf{SSL-mac}}$. In contrast to the TLS-combiner, SSL uses the entire master secret as key for both hash functions. This approach results in a construction $\mathsf{Comb}^{H_0,H_1}_{\mathsf{SSL-mac}}$ that is not even MAC-preserving, although concatenation is MAC-robust when used with distinct keys for each hash function [Her05].

Proposition 4.3 *The combiner* $\mathsf{Comb}^{H_0,H_1}_{\mathsf{SSL-mac}}$ *is* not *MAC-preserving (and thus* not *MAC-robust either).*

Consider two secure MACs $\mathsf{H}_0, \mathsf{H}_1$, that on input of a secret key k and a message M outputs a tag σ_b. Assume furthermore that both MACs ignore parts of their key, i.e., H_0 ignores the left half of its input key and H_1 ignores the right part. We now derive functions H^*_b that can still be unforgeable when used alone, but become totally insecure when being plugged into the combiner. The first MAC H^*_0 behaves like H_0 but also leaks the left half k_l of the secret key, i.e., $\mathsf{H}^*_0(k, M) = k_l||\mathsf{H}_0(k, M)$. The second function is defined analogously, but outputs the right half of the key: $\mathsf{H}^*_1(k, M) = k_r||\mathsf{H}_1(k, M)$. Even though each tag is now accompanied with a part of the key, it remains hard to create a forgery. When we use now both functions $\mathsf{H}^*_0, \mathsf{H}^*_1$ as in the SSL-combiner[3] we obtain: $\mathsf{H}^*_0(k, M) \;||\; \mathsf{H}^*_1(k, M) = k_l||\sigma_0||k_r||\sigma_1$ which allows to easily reconstruct the entire secret key and subsequently forge tags for any message.

Improvements. In order to change the SSL-construction such that it becomes MAC-robust, the key should be split among both underlying hash functions. The concatenation of $\mathsf{HMAC}^*_{H_0}, \mathsf{HMAC}^*_{H_1}$ invoked with independent keys then clearly gives a secure MAC, if at least one of the underlying functions is unforgeable. As SSL deviates from the standard HMAC approach to build its MAC algorithms, the results from Section 4.1 do not apply for HMAC^*. However, Dodis and Puniya scrutinized in [DP08] the minimal assumptions of a compression function such that the corresponding iterated hash function (with various keying approaches) yields a secure MAC. They showed that a MAC based on the Merkle-Damgård construction with the key appended to the message, requires the compression function to be collision-resistant and unforgeable when it gets keyed by the input data. It is also claimed that HMAC with an appended key requires the same assumptions as the plain MD approach. Thus, the outer hash application in $\mathsf{HMAC}^*_{H_b}$ does not contribute to the security of the MAC, in the sense that it relaxes the underlying assumption and therefore can be omitted. Applying both discussed modifications we obtain the following construction:

$$\mathsf{Comb}^{H_0,H_1}_{\mathsf{SSL-mac}^*}(k_0||k_1, M) = H_0(M||k_0) \;||\; H_1(M||k_1)$$

If it is desirable to save on communication, one can use the xor of the H_0, H_1 outputs instead of the concatenation, while retaining the same security guarantees: In both cases the proposed combiner is a robust MAC if at least one compression function is *simultaneously* collision-resistant and unforgeable. Note that this is a stronger assumption than for the TLS combiner, where both properties can be possessed by possibly different functions.

[3]Invoking the combiner directly on H^*_b instead of $\mathsf{HMAC}^*_{\mathsf{H}^*_b}$ still proves our statement as the HMAC transform can inherit the behavior H^*. We omit the additional level for the sake of simplicity.

Acknowledgments

We thank the anonymous reviewers for valuable comments. The first two authors are supported by the Emmy Noether Program Fi 940/2-1 of the German Research Foundation (DFG).

References

[AB99] Jee Hea An and Mihir Bellare. *Constructing VIL-MACsfrom FIL-MACs: Message Authentication under Weakened Assumptions.* Advances in Cryptology — Crypto 99, Volume 1666 of LNCS, pages 252–269. Springer, 1999.

[BB06] Dan Boneh and Xavier Boyen. *On the Impossibility of Efficiently Combining Collision Resistant Hash Functions.* Advances in Cryptology — Crypto 2006, Volume 4117 of LNCS, pages 570–583. Springer, 2006.

[BCK96a] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. *Keying hash functions for message authentication.* Advances in Cryptology — Crypto 1996, Volume 96 of LNCS, pages 1–15. Springer, 1996.

[BCK96b] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. *Pseudorandom Functions Revisited: The Cascade Construction and Its Concrete Security.* Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS) 1996, pages 514–523. IEEE Computer Society Press, 1996.

[Bel06] Mihir Bellare. *New Proofs for NMAC and HMAC: Security without Collision-Resistance.* Advances in Cryptology — Crypto 2006, Volume 4117 of LNCS, pages 602–619. Springer, 2006.

[CR08] Christophe De Cannière and Christian Rechberger. *Preimages for Reduced SHA-0 and SHA-1.* Advances in Cryptology — Crypto 2008, Volume 5157 of LNCS, pages 179–202. Springer, 2008.

[CRS+07] Ran Canetti, Ronald L. Rivest, Madhu Sudan, Luca Trevisan, Salil P. Vadhan, and Hoeteck Wee. *Amplifying Collision Resistance: A Complexity-Theoretic Treatment.* Advances in Cryptology — Crypto 2007, Volume 4622 of LNCS, pages 264–283. Springer, 2007.

[Dam90] Ivan Damgård. *A Design Principle for Hash Functions.* Advances in Cryptology — Crypto 1989, Volume 435 of LNCS, pages 416–427. Springer, 1990.

[DP08] Yevgeniy Dodis and Prashant Puniya. *Getting the Best Out of Existing Hash Functions; or What if We Are Stuck with SHA?* Applied Cryptography and Network Security (ACNS) 2008, Volume 5037 of LNCS, pages 156–173. Springer, 2008.

[Fis08] Marc Fischlin. *Security of NMAC and HMAC Based on Non-malleability.* Topics in Cryptology — Cryptographer's Track, RSA Conference (CT-RSA) 2008, Volume 4964 of LNCS, pages 138–154. Springer, 2008.

[FL07] Marc Fischlin and Anja Lehmann. *Security-Amplifying Combiners for Hash Functions.* Advances in Cryptology — Crypto 2007, Volume 4622 of LNCS, pages 224–243. Springer, 2007.

[FL08] Marc Fischlin and Anja Lehmann. *Robust Multi-Property Combiners for Hash Functions.* Theory of Cryptography Conference (TCC) 2008, Volume 4948 of LNCS, pages 375–392. Springer, 2008.

[FL10] Marc Fischlin and Anja Lehmann. *Delayed-Key Message Authentication for Streams.* Theory of Cryptography Conference (TCC) 2010, LNCS. Springer, 2010.

[FLP08] Marc Fischlin, Anja Lehmann, and Krzysztof Pietrzak. *Robust Multi-Property Combiners for Hash Functions Revisited.* International Colloquium on Automata, Languages, and Progamming (ICALP) 2008, Volume 5126 of LNCS, pages 655–666. Springer, 2008.

[GMP+08] Sebastian Gajek, Mark Manulis, Olivier Pereira, Ahmad-Reza Sadeghi, and Jörg Schwenk. *Universally Composable Security Analysis of TLS.* ProvSec 2008, Volume 5324 of LNCS, pages 313–327. Springer, 2008.

[Her05] Amir Herzberg. *On Tolerant Cryptographic Constructions.* Topics in Cryptology — Cryptographer's Track, RSA Conference (CT-RSA) 2005, Volume 3376 of LNCS, pages 172–190. Springer, 2005.

[Hir04] Shoichi Hirose. *A note on the strength of weak collision resistance. IEICE Transactions on fundamentals of electronics communications and computer sciences*, 87(5):1092–1097, 2004.

[KL08] Jonathan Katz and Andrew Y. Lindell. *Aggregate Message Authentication Codes.* Topics in Cryptology — Cryptographer's Track, RSA Conference (CT-RSA)'08, LNCS, pages 155–169. Springer, 2008.

[Kra08] Hugo Krawczyk. *On Extract-then-Expand Key Derivation Functions and an HMAC-based KDF.* `http://webee.technion.ac.il/ hugo/kdf/kdf.pdf`, 2008.

[Mer90] Ralph Merkle. *One Way Hash Functions and DES.* Advances in Cryptology — Crypto 1989, Volume 435 of LNCS, pages 428–446. Springer, 1990.

[MSW08] Paul Morrissey, Nigel Smart, and Bogdan Warinschi. *A Modular Security Analysis of the TLS Handshake Protocol.* Advances in Cryptology — Asiacrypt 2008, Volume 5350 of LNCS, pages 55–73. Springer, 2008.

[Pie07] Krzysztof Pietrzak. *Non-Trivial Black-Box Combiners for Collision-Resistant Hash-Functions don't Exist.* Advances in Cryptology — Eurocrypt 2007, Volume 4515 of LNCS, pages 23–33. Springer, 2007.

[Pie08] Krzysztof Pietrzak. *Compression from Collisions, or why CRHF Combiners have a Long Output.* Advances in Cryptology — Crypto 2008, Volume 5157 of LNCS, pages 413–432. Springer, 2008.

[Res01] Eric Rescorla. *SSL and TLS - Designing and Building Secure Systems.* Addison Wesley, 2001.

[Rog06] Phillip Rogaway. *Formalizing Human Ignorance.* Vietcrypt 2006, Volume 4341 of LNCS, pages 211–228. Springer, 2006.

[SSA+09] Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger. *Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate.* Advances in Cryptology — Crypto 2009, Volume 5677 of LNCS, pages 55–73. Springer, 2009.

[SSL94] *The SSL Protocol (Internet Draft).* Technical report, K.E.B. Hickman, 1994.

[TLS99] *The TLS Protocol Version 1.0.* Technical Report RFC 2246, T. Dierks, and C. Allen, 1999.

[TLS06] *The TLS Protocol Version 1.2.* Technical Report (TLS 1.2) RFC 4346, T. Dierks, and C. Allen, 2006.

[WY05] Xiaoyun Wang and Hongbo Yu. *How to break MD5 and other hash functions.* Advances in Cryptology — Eurocrypt 2005, Volume 3494 of LNCS, pages 19–35. Springer, 2005.

[WYY05] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. *Finding collisions in the full SHA-1.* Advances in Cryptology — Crypto 2005, Volume 3621 of LNCS, pages 17–36. Springer, 2005.

A PRF-Combiner based on Insecure Hash Functions

In this section we discuss that even functions that in general exhibit no random behavior, can yield a pseudorandom function when used in the TLS or SSL combiner. For both examples we start from secure PRFs $\mathsf{H}_0, \mathsf{H}_1$ which we modify into functions $\mathsf{H}_0^*, \mathsf{H}_1^*$ that lose their security when used in a stand-alone fashion.

Example for the SSL combiner $\mathsf{Comb}^{H_0,H_1}_{\mathsf{SSL-prf}}$ Our example for SSL holds for the weaker version of the combiner where both functions get keyed with the same master secret, and which we showed to be neither PRF-robust nor preserving in general. Consider the function $\mathsf{H}^*_1(k,m) = k||m$ that simply outputs its secret key and the message it was invoked on. This function is clearly not pseudorandom. The second function $\mathsf{H}^*_0(k,m)$ parses each input as $m = k'||m'$ and checks whether $k = k'$ holds. If so it outputs $\mathsf{H}_0(k,m)$, else 0^n. When H^*_0 is used alone, the probability of hitting an input among q queries, whose prefix matches the secret key $k \leftarrow \{0,1\}^n$, is $q \cdot 2^{-n}$ and thus negligible. Hence, with overwhelming probability one merely gets replies 0^n and can thus easily distinguish this function from random. However, in the combination $\mathsf{H}^*_0(k, \mathsf{H}^*_1(k,m))$, the outer function will always run in the "good" exceptional state where it behaves like H_0 and provides random values. Thus, we exploit the same weakness as in 3.2 where we showed that the SSL combiner is not even PRF-preserving, but now use that peculiarity to obtain a secure PRF out of insecure functions.

Example for the TLS combiner $\mathsf{Comb}^{H_0,H_1}_{\mathsf{TLS-prf}}$ For TLS we can obtain a similar result by constructing the following functions $\mathsf{H}^*_0(k,m) = 0^{n/2}||H_0(k,m)|_{n/2}$ and $\mathsf{H}^*_1(k,m) = H_1(k,m)|_{n/2}||0^{n/2}$, where $x|_{n/2}$ denotes the leading $n/2$ bits of string x. Both functions output strings that consist of a constant half and a random half, and are obviously easily distinguishable from a truly random function. By merging them into the TLS combiner, we obtain $\mathsf{H}^*_0(k_0,m) \oplus \mathsf{H}^*_1(k_1,m)$ which nullifies the constant parts and yields random strings again.