Delayed-Key Message Authentication for Streams

Marc Fischlin and Anja Lehmann

Darmstadt University of Technology, Germany
www.minicrypt.de

Abstract. We consider message authentication codes for streams where the key becomes known only at the end of the stream. This usually happens in key-exchange protocols like SSL and TLS where the exchange phase concludes by sending a MAC for the previous transcript and the newly derived key. SSL and TLS provide tailor-made solutions for this problem (modifying HMAC to insert the key only at the end, as in SSL, or using upstream hashing as in TLS). Here we take a formal approach to this problem of delayed-key MACs and provide solutions which are "as secure as schemes where the key would be available right away" but still allow to compute the MACs online even if the key becomes known only later.

1 Introduction

With the final step in key exchange protocols the parties usually authenticate the previous communication. This is typically achieved by exchanging message authentication codes $\mathsf{Mac}(\mathsf{K}, \mathsf{transcript})$ computed over the transcript of the communication. Examples include the final message in the handshake protocol of SSL and TLS [17], as well as many other key exchange protocols [4,13,14,8].

The intriguing observation here is that the key for the MAC computations becomes only known after the transcript is provided. We call this *delayed-key* authentication. For such schemes, even MACs which potentially allow to authenticate streams may need to store the entire transcript before the MAC can be derived. One well-known example is HMAC where the (inner) key is prepended to the message before hashing, $H(\mathsf{K}_{\text{out}}, H(\mathsf{K}_{\text{in}}, m))$. In this case the key must be available *before* processing the message in order to take advantage of the iterated hash function structure.

For computational efficiency and, especially, for storage reasons it is often desirable to compute the MAC iteratively, though. This has been acknowledged by popular protocols like SSL, which uses a variant of HMAC where the key is *appended* to the message instead, and TLS which first hashes the transcript iteratively and then runs the MAC on the hash value only. Similarly, for the key exchange protocols for machine readable travel documents (MRTD) by the German government [5] the final MAC computation omits large parts of the

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-3-642-11799-2_36

D. Micciancio (Ed.): TCC 2010, LNCS 5978, pp. 290–307, 2010.

transcript and only inputs the messages of the final rounds. This allows the resource-bounded passport to free memory immediately. The protocol is under standardization for ISO/IEC JTC1/SC17.

The SSL and TLS solution to the problem both rely on the collision resistance of the underlying hash function for HMAC.[1] For TLS collision resistance suffices to show security (assuming HMAC is secure), but introduces another requirement on the hash function. Recall that HMAC (resp. its theoretical counterpart NMAC) can be shown to be secure if the compression function is pseudorandom [1] or non-malleable [7]. For SSL it is still unclear how the security of the modified HMAC relates to the security of the original HMAC. As for the MRTD protocol for German passports, in most key exchange protocols it is recommended to include the whole transcript (yet, we are not aware of any concrete attack if only parts of the transcript enter the computation).

An additional constraint originates from the implementation of the MAC algorithm. Key-exchange protocols are often used as building blocks in more complex cryptographic protocols which, in turn, also use the same MAC algorithm for subsequent authentication (e.g., the record protocol in TLS/SSL). To be applicable to resource-bounded devices a delayed-key MAC should therefore draw on the same implementation as the regular MAC. This is particularly true if the implementation has been designed to resist side-channel attacks. Hence, instead of designing delayed-key MACs from scratch, a "lightweight" transformation given an arbitrary MAC algorithm is preferable.

Our Results. We initiate a study of solutions for the delayed-key MAC problem. There are two reasonable scenarios, originating from the key-exchange application: The most relevant case in practice is the *one-sided* case where one party is resource-bounded while the other party is more powerful, e.g., a TLS/SSL secured connection between a mobile device and a server, or an authentication procedure between a smart card and a card reader. Then, ideally, the constraint device should benefit from solutions with low storage, whereas we can still assume that the server is able to store the entire transcript. If both parties have storage limitations, e.g., two mobile devices communicating with each other, then we are interested in *two-sided* solutions. Since the one-sided case allows for the weaker devices in terms of resource constraints, the necessity of storage-optimized protocols in this scenario is usually higher than in the two-sided case.

Thus, we focus on the one-sided case for which we present efficient solutions which are all based on the same seemingly obvious principle: to compute a MAC the sending party first picks an ephemeral key L and computes the MAC for this key and the data stream. Then, in addition to the MAC under this key, the party also transmits an "encryption" (or a "pointer") P allowing the other party to recover the ephemeral key L from P and the meanwhile available long-term

[1] The weaker requirement of preimage resistance does not suffice, because the transcript that gets authenticated, is partially determined by both the sender and the receiver of the MAC.

key K.[2] Note that since verification is usually done by re-computing a MAC the idea also applies to the verification of the other's party MAC, i.e., one of the parties in a key-exchange protocol can both compute its own MAC and verify the other party's MAC with low storage requirements.

From an efficiency and implementation viewpoint the instantiations of this principle should interfere as little as possible with the underlying protocol such that we get a universal solution. Note that this general approach already allows to obtain a delayed-key solution starting from a regular MAC, such that both variants can be used conveniently even on severely constraint devices. In terms of security we require the solution to be as secure as the original scheme. The latter condition at foremost demands that the instantiation inherits the unforgeability property of the original MAC. But since the long-term key K is subsequently used in protocols (like encryption with the derived keys from the master secret in SSL and TLS), unforgeability alone is not sufficient.

We also demand that the modified scheme only leaks "as much about the key K as the original scheme would" and call this notion *leakage-invariance*. The idea behind this notion is that, in the original key-exchange protocol, the MAC for K leaks some information about the key itself, and that the subsequent usage of the key (derivation, direct encryption etc.) should be still be secure. Following the idea of semantically secure encryption [10] we require that a solution for the delayed-key problem allows to compute at most the information about K that one could derive from a $\mathsf{Mac}(\mathsf{K},\cdot)$ oracle (used in the original protocol).

We discuss four solutions which are secure according to our notion (and which come with different efficiency/security trade-offs). Roughly, these are:

Encrypt-then-MAC: We assume that the underlying (deterministic) MAC is a pseudorandom function (which is a widely used assumption about HMAC) and then compute the MAC $\sigma \leftarrow \mathsf{Mac}(\mathsf{L}, m, \ell)$ for the ephemeral key and then encrypt L under K and MAC this data, $\mathsf{P} = (c,t) = (\mathsf{Mac}(\mathsf{K}, 0||\ell) \oplus \mathsf{L}, \mathsf{Mac}(\mathsf{K}, 1||\ell||c))$ for a label ℓ which can either be the server or client constant as in SSL or a random session identifier. The receiver can then recover L from the encryption and verify the MAC σ.

Pseudorandom Permutation: We again assume that the MAC is a pseudorandom function and use a four-round Feistel structure to build a pseudorandom permutation $\pi(\mathsf{K},\cdot)$ out of it. Then $\sigma \leftarrow \mathsf{Mac}(\mathsf{L}, m)$ and $\mathsf{P} = \pi^{-1}(\mathsf{K}, \mathsf{L})$ such that the receiver can re-obtain $\mathsf{L} = \pi(\mathsf{K}, \mathsf{P})$ and verify the MAC σ. The communication overhead here is smaller than in the previous case but the construction requires more MAC computations.

Encrypt-only: For the pseudorandom MAC we simply let $\mathsf{P} = (\ell, \mathsf{Mac}(\mathsf{K}, \ell) \oplus \mathsf{L})$ for random label ℓ. In this case the security condition is that an adversary

[2] This approach is more general than it may seem at first glance: One can think of the MAC computation for key L as a (probabilistic) processing of the message and the final computation of the pointer (from K, L and the value from the first stage) as an "enveloping" transformation involving the key. It comprises for example the SSL/TLS solutions (with empty L). We finally remark that sending L in clear usually violates the secure deployment of such MACs in key agreement protocols.

attacking this modified scheme can only make a limited number of verification requests (which corresponds to the common case that in two-party key-exchange protocols for each exchanged key K the server and the client compute and verify only one MAC each). Also, we can only show that the adversary is unable to recover the entire key K from the modified scheme (in contrast to any information about the key, as in the previous cases). This is sufficient to provide security if the key is afterwards hashed (assuming that the hash functions is a good randomness extractor or even behaves like a random oracle).

XOR: In the most simple case we let $\mathsf{P} = \mathsf{K} \oplus \mathsf{L}$ be the one-time pad encryption of L under K. Assuming that MAC remains pseudorandom under related-key attacks [3] this is again an unforgeable, leakage-invariant MAC (if the adversary task is to recover the whole key K). The leakage-invariance also relies on the assumption that the adversary can only make a limited number of verification queries, and gets to see at most one MAC. The latter is justified in schemes where only one of the party sends a MAC or where one party immediately aborts without sending its MAC if the received MAC is invalid.

As mentioned before all proposed solutions above support the one-sided case where one of the parties can store the message easily. In contrast, the TLS/SSL solutions also work in the two-sided case of two resource-constraint parties, but both rely on the collision-resistance of the underlying hash function whereas our solutions can in principle be implemented based on one-way functions. We therefore address the question whether or not collision-resistance is necessary for the two-sided case or not, and show that one-way functions suffice. However, as our solution make use of digital signatures it is mainly a proof of concept and it remains an interesting open problem to find more efficient constructions for this case.

Related Results. To the best of our knowledge the delayed-key problem has not undergone a comprehensive formal treatment so far. The solution in TLS can be shown to be secure according to our model, but relies on collision-resistance. As attacks have shown, however, this appears to be a stronger assumption than pseudorandomness, especially in light of the deployed hash functions MD5 and SHA-1 in TLS (see also the discussion in [1]). We note that relaxing the requirement of collision-resistance is also a goal in other areas like hash-and-sign schemes [12].

Closest to our setting here comes the scenario of broadcast authentication of streams via the TESLA protocol [16]. There, the two parties share a one-way chain of keys and authenticate each packet in time t with the t-th key of the chain. Hence, TESLA also deals with authentication of streams and supports limited buffering, but in contrast to our setting TESLA covers immediate authentication of packets, requiring synchronization between the parties, and assumes shared keys right away (whereas our key is delayed).

Analogously to TESLA, all other works on stream authentication refer to immediate verification of each packet, e.g. [11].

In a recent work, Garay et al. [9] also address the problem of MAC precomputations. However, they consider MACs in the context of hardware security and show how to perform most of a MAC computation offline, before the message is available.

2 Preliminaries

In this section we introduce the basic notions for message authentication codes. In the key exchange application the two parties at the end usually compute the MAC for the same message m but include their identity in the message. For instance, SSL includes the server and client constant in the computation of the finished message. Alternatively, the label can also be a random value chosen by the party computing the MAC. In any case we assume that the label is known at the outset of the MAC computation. We thus introduce labels in the model such that each message m is escorted by a label $\ell \in \{0,1\}^n$ and the authentication code covers both parts. We note that, for regular MACs, this is rather a syntactic modification and becomes important only for the case of delayed-key MACs.

Definition 1. *A message authentication code scheme* $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vf})$ *(with labels) is a triple of efficient algorithms where*

Key Generation. $\mathsf{KGen}(1^n)$ *gets as input the security parameter* 1^n *and returns a key* k*.*

Authentication. *The authentication algorithm* $\sigma \leftarrow \mathsf{Mac}(k, m, \ell)$ *takes as input the key* k*, a message* m *from a space* $\mathcal{M}_n$ *and a label* $\ell \in \{0,1\}^n$ *and returns a tag* σ *in a range* $\mathcal{R}_n$*.*

Verification. $\mathsf{Vf}(k, m, \ell, \sigma)$ *returns a bit.*

It is assumed that the scheme is complete*, i.e., for all* $k \leftarrow \mathsf{KGen}(1^n)$*, any* $(m, \ell) \in \mathcal{M}_n$*, and any* $\sigma \leftarrow \mathsf{Mac}(k, m, \ell)$ *we have* $\mathsf{Vf}(k, m, \ell, \sigma) = 1$*.*

A MAC is called deterministic if algorithm Mac is deterministic. Unforgeability of MACs demands that it is infeasible to produce a valid tag for a new message:

Definition 2. *A message authentication code* $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vf})$ *(with labels) is called* unforgeable under chosen message attacks *if for any efficient algorithm* $\mathcal{A}$ *the probability that the experiment* $\mathsf{Forge}_{\mathcal{A}}^{\mathsf{MAC}}$ *evaluates to* 1 *is negligible (as a function of* n*), where*

Experiment $\mathsf{Forge}_{\mathcal{A}}^{\mathsf{MAC}}(n)$
$k \leftarrow \mathsf{KGen}(1^n)$
$(m^*, \ell^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{MAC}(k,\cdot,\cdot),\mathsf{Vf}(k,\cdot,\cdot,\cdot)}(1^n)$
Return 1 *iff*
$\mathsf{Vf}(k, m^*, \ell^*, \sigma^*) = 1$ *and* $\mathcal{A}$ *has never queried* $\mathsf{Mac}(k, \cdot, \cdot)$ *about* (m^*, ℓ^*)*.*

Note that for deterministic MACs where, in addition, the verification algorithm recomputes the tag and compares it to the given tag, the verification oracle $\mathsf{Vf}(k, \cdot, \cdot, \cdot)$ can be omitted [2] while decreasing the adversary's success probability by at most the number of verification queries. This particularly holds for HMAC.

For some of our security proofs it is necessary to assume that the MAC is a pseudorandom function. We note again that HMAC (or, to be precise, NMAC) has this property as long as the underlying compression function is pseudorandom [1].

Definition 3. *A message authentication code* MAC *is a* pseudorandom function *if for any efficient distinguisher* $\mathcal{D}$ *the advantage*

$$\left|\text{Prob}\left[\mathcal{D}^{\mathsf{Mac}(k,\cdot)}(1^n)=1\right]-\text{Prob}\left[\mathcal{D}^{f(\cdot)}(1^n)=1\right]\right|$$

is negligible, where the probability in the first case is over $\mathcal{D}$*'s coin tosses and the choice of* $k \leftarrow \mathsf{KGen}(1^n)$*, and in the second case over* $\mathcal{D}$*'s coin tosses and the choice of the random function* $f : \mathcal{M}_n \to \mathcal{R}_n$*.*

3 Defining Delayed-Key MACs for Streams

As explained in the introduction in the setting of MACs for streams where the key K is only available at the end of the communication, we augment the MAC by a function Point which maps the ephemeral key L (used to derive the MAC for the stream) via K to a pointer P, and such that the verifier can recover the ephemeral key from this pointer and K by the "inverse" Point^{-1}. We let Point also depend on the MAC σ computed with the ephemeral key to capture general solutions as in TLS and since this information is available when computing the pointer (see also the remark after the definition). If Point does not depend on σ we usually omit it from the algorithm's input.

Definition 4. *A delayed-key message authentication code scheme* $\mathsf{DKMAC} = (\mathsf{KGen}, (\mathsf{Mac}, \mathsf{Point}), \mathsf{Vf})$ *(with labels) is a tuple of efficient algorithms where*

Key Generation. $\mathsf{KGen}(1^n)$ *gets as input the security parameter* 1^n *and returns a secret key* K*.*

Authentication. *Algorithm* Mac *on input an ephemeral key* L*, a message* m *and a label* ℓ *returns a tag* σ*, and algorithm* Point *for input two keys* K *and* L *and the label* ℓ *returns a pointer* P*. An augmented tag for key* K *and* (m, ℓ) *then consists of the pair* $(\sigma, \mathsf{P}) \leftarrow (\mathsf{Mac}(\mathsf{L}, m, \ell), \mathsf{Point}(\mathsf{K}, \mathsf{L}, \ell, \sigma))$ *for random* $\mathsf{L} \xleftarrow{\$} \mathsf{KGen}(1^n)$*.*

Verification. $\mathsf{Vf}(\mathsf{K}, \mathsf{P}, m, \ell, \sigma)$ *returns a bit.*

It is assumed that the scheme is complete*, i.e., for any* $\mathsf{K} \leftarrow \mathsf{KGen}(1^n)$*, any* $(m, \ell) \in \mathcal{M}_n \times \{0,1\}^n$*, any augmented tag* $(\sigma, \mathsf{P}) \leftarrow (\mathsf{Mac}(\mathsf{L}, m, \ell), \mathsf{Point}(\mathsf{K}, \mathsf{L}, \ell))$ *for* $\mathsf{L} \leftarrow \mathsf{KGen}(1^n)$ *we have* $\mathsf{Vf}(\mathsf{K}, \mathsf{P}, m, \ell, \sigma) = 1$*.*

Both the SSL as well as the TLS solution can be mapped trivially to the definition above. Namely, in both cases the ephemeral key L is the empty string and the "MAC" σ is merely the hash value of the message. The pointer P is then the result of the actual MAC computations for K (i.e., HMAC with appended key in SSL and HMAC for the hash value in TLS).

We remark that in key exchange protocols usually both parties send a MAC of the transcript, possibly adding some distinct public identifiers. Our notion of delayed-key MACs can be easily used to model the one-sided case with a bounded client and a powerful server such that the client can *compute* its own MAC and *verify* the server's MAC with limited storage only (assuming that the

underlying MAC implements verification by recomputing the MAC and comparing the outcome to the given tag): Namely, the client uses an ephemeral key L to compute its own MAC, and another ephemeral key L' to start computing the server's MAC for verification. At the end, the client transmits the pointers P and P' for the two MACs and the server derives L, L' through K and verifies the client MAC and computes and sends its own MAC. The client then only needs to verify that this received MAC matches the previously computed value.

3.1 Security of Delayed-Key MACs

We adapt the security requirement of unforgeable MACs to our scenario of delayed-key MACs, i.e., we grant the adversary access to an oracle $\mathcal{O}_{\mathsf{MAC}}(\mathsf{K}, \cdot)$ that is initialized with a secret key K and mimics the authentication process, returning augmented tags. Thus, for every query the oracle first chooses a fresh ephemeral key L_i and then returns the augmented tag $(\sigma_i, \mathsf{P}_i) \leftarrow (\mathsf{Mac}(\mathsf{L}_i, m_i, \ell_i), \mathsf{Point}(\mathsf{K}, \mathsf{L}_i, \ell_i, \sigma_i,))$. After learning several tags the adversary eventually halts and outputs a tuple $(\mathsf{P}^*, m^*, \ell^*, \sigma^*)$. The adversary is successful if the output verifies as true under key K and the oracle has never been invoked on (m^*, ℓ^*).

Definition 5. *A delayed-key message authentication code* $\mathsf{DKMAC} = (\mathsf{KGen}, (\mathsf{Mac}, \mathsf{Point}), \mathsf{Vf})$ *(with labels) is called* unforgeable under chosen message attacks *if for any efficient algorithm* $\mathcal{A}$ *the probability that the experiment* $\mathsf{Forge}_{\mathcal{A}}^{\mathsf{DKMAC}}$ *evaluates to* 1 *is negligible (as a function of* n*), where*

Experiment $\mathsf{Forge}_{\mathcal{A}}^{\mathsf{DKMAC}}(n)$
$\mathsf{K} \leftarrow \mathsf{KGen}(1^n)$
$(\mathsf{P}^*, m^*, \ell^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{MAC}}(\mathsf{K}, \cdot)}(1^n)$
where $\mathcal{O}_{\mathsf{MAC}}(\mathsf{K}, \cdot)$ *for every query* (m_i, ℓ_i) *samples a fresh* $\mathsf{L}_i \leftarrow \mathsf{KGen}$
and returns $(\sigma_i, \mathsf{P}_i) \leftarrow (\mathsf{Mac}(\mathsf{L}_i, m_i, \ell_i, \sigma_i), \mathsf{Point}(\mathsf{K}, \mathsf{L}_i, \ell_i))$
Return 1 *iff*
$\mathsf{Vf}(\mathsf{K}, \mathsf{P}^*, m^*, \ell^*, \sigma^*) = 1$
and $\mathcal{A}$ *has never queried* $\mathcal{O}_{\mathsf{MAC}}(\mathsf{K}, \cdot)$ *about* (m^*, ℓ^*).

When a MAC is used in a stand-alone fashion the security guarantee of unforgeability usually suffices. However, when applied as a building block in protocols like TLS or SSL the MAC is computed for a key which is subsequently used to derive further keys or to encrypt data. Besides the regular unforgeability requirement it is thus also necessary to ensure that any delayed-key MAC is "as secure as applying the original MAC". That is, the delayed-key MAC should leak at most the information about the key K as the deployment of the original MAC does.

We therefore introduce the notion of *leakage-invariance*, basically saying that MACs may leak information about the key, but this information does not depend on the specific key value. In our setting this means that the leakage of the ephemeral keys and of the long-term key for each MAC computation are identical (yet, since we augment the tag by the pointer we still need to ensure that this extra information does not violate security). More formally, we compare the

success probability of an adversary $\mathcal{A}$ predicting some information $f(\mathsf{K})$ about key K after learning several tuples $(\mathsf{P}_i, m_i, \ell_i, \sigma_i)$ with the success probability of an adversary $\mathcal{B}$ given only access to the plain underlying authentication algorithm $\mathsf{Mac}(\mathsf{K}, \cdot, \cdot)$. For a leakage-invariant delayed-key MAC these probabilities should be close.

Definition 6. *A delayed-key* $\mathsf{DKMAC} = (\mathsf{KGen}, (\mathsf{Mac}, \mathsf{Point}), \mathsf{Vf})$ *(with labels) is called* leakage-invariant *if for any probabilistic polynomial-time algorithm* $\mathcal{A}$ *there exists a probabilistic polynomial-time algorithm* $\mathcal{B}$ *such that for any (probabilistic) function* f *the difference*

$$\mathrm{Prob}\Big[\,\boldsymbol{Exp}^{\textit{leak-inv}}_{\mathcal{A},\mathsf{DKMAC}}(n) = 1\Big] - \mathrm{Prob}\Big[\,\boldsymbol{Exp}^{\textit{leak-inv}}_{\mathcal{B},\mathsf{DKMAC}}(n) = 1\Big]$$

is negligible, where:

Experiment $\boldsymbol{Exp}^{\textit{leak-inv}}_{\mathcal{A},\mathsf{DKMAC}}(n)$	***Experiment*** $\boldsymbol{Exp}^{\textit{leak-inv}}_{\mathcal{B},\mathsf{DKMAC}}(n)$
$\mathsf{K} \leftarrow \mathsf{KGen}(1^n)$	$\mathsf{K} \leftarrow \mathsf{KGen}(1^n)$
$a \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{MAC}}(\mathsf{K},\cdot),\mathsf{Vf}(\mathsf{K},\cdots)}(1^n)$	$a \leftarrow \mathcal{B}^{\mathsf{Mac}(\mathsf{K},\cdot,\cdot),\mathsf{Vf}(\mathsf{K},\cdot,\cdot)}(1^n)$
where $\mathcal{O}_{\mathsf{MAC}}(\mathsf{K}, m_i)$ *samples a key* $\mathsf{L}_i \leftarrow \mathsf{KGen}(1^n)$ *and returns* $(\sigma_i, \mathsf{P}_i) \leftarrow (\mathsf{Mac}(\mathsf{L}_i, m_i, \ell_i), \mathsf{Point}(\mathsf{K}, \mathsf{L}_i, \ell_i, \sigma_i))$	
output 1 if and only if $a = f(\mathsf{K})$	*output 1 if and only if* $a = f(\mathsf{K})$

If the function f *is from a set* $\mathcal{F}$ *of functions and* $\mathcal{A}$ *makes at most* q_{Mac} *queries to oracle* $\mathcal{O}_{\mathsf{MAC}}$ *and at most* q_{Vf} *queries to oracle* Vf*, then we say that the MAC is* $(q_{\mathsf{Mac}}, q_{\mathsf{Vf}}, \mathcal{F})$-leakage-invariant*. The scheme is called* leakage-invariant for distinct labels *if* $\mathcal{A}$ *only submits queries with distinct labels to oracle* $\mathcal{O}_{\mathsf{MAC}}(\mathsf{K}, \cdot, \cdot)$*. It is called* leakage-invariant for random labels *if the labels are chosen at random by oracle* $\mathcal{O}_{\mathsf{MAC}}$ *(instead of being picked by the adversary).*

We can even strengthen our definition by bounding the adversary $\mathcal{B}$ to the number of $\mathcal{A}$'s queries, i.e., if $\mathcal{A}$ can derive some information $f(\mathsf{K})$ in $q = (q_{\mathsf{Mac}}, q_{\mathsf{Vf}})$ queries, then $\mathcal{B}$ should be able to deduce $f(\mathsf{K})$ in at most q queries as well. We call such schemes *strongly leakage-invariant*. We do not impose such a restriction per se, since there can be leakage-invariant solutions where $\mathcal{B}$ can safely make more queries (e.g., if MACs are pseudorandom, except that they always leak the first three bits of the key).

Above we do not put any restriction on the function f, i.e., it could even be not efficiently computable. For our more efficient solution we weaken the notion above and demand that the adversary computes the identity function $f(\mathsf{K}) = \mathsf{K}$, i.e., predicts the entire key. Formally, we then let $\mathcal{F} = \{\mathrm{ID}\}$. If, as done in most key exchange protocols, the key is subsequently piped through a hash function modeled as a random oracle, then the adversary needs to query the random oracle about the entire key (and thus needs to predict it). Else the adversary is completely oblivious about the random hash value and the derived key. In other words, in this scenario considering the identity function suffices.

We remark that we refrain from using Canetti's universal composition (UC) model [6] although we are interested in how the key is subsequently used. The second experiment with adversary $\mathcal{B}$ of our notion of leakage-invariance already resembles the notion of an ideal functionality and the ideal-world scenario, and the actual attack on the concrete scheme mimics the real-world setting. However, the UC model introduces additional complications like session IDs and seems to provide more than what is often needed in the applications we have in mind (i.e., one typically asks for more than that the adversary cannot recover the entire key, even though this may be sufficient).

We finally note that the "TLS solution" to first compute $H(m)$ and then $\mathsf{Mac}(\mathsf{K}, H(m), \ell)$ is clearly strongly leakage-invariant if H is collision-resistant (essentially because the ephemeral key L is empty, $\sigma_i = H(m_i)$ is publicly known and the pointer P is the MAC for σ_i). In addition, it is also unforgeable, providing a secure solution under the stronger assumption.

Leakage-Invariance vs. Unforgeability. In general, the notions of unforgeability and leakage-invariance are somewhat incomparable, as we show by separating examples in the full version of the paper. However, in the case that the leakage invariance is limited to the function $f = \mathrm{ID}$ which is the prediction of the entire key, an adversary against leakage-invariance trivially gives an adversary against the unforgeability, as well.

4 One-Sided Delayed-Key MACs: The Unbounded Case

In this section we present our first construction of a delayed-key MAC, that uses a pseudorandom MAC as building block. We show that this approach is unforgeable and leakage-invariant if the underlying MAC is a pseudorandom function. This is independent of any bound on the number of MAC or verification queries and of any assumption about the function f. We present our second construction for the unbounded case in the full version of the paper.

4.1 Pseudorandom Permutation

The idea of our construction $\mathsf{DKMAC}_{\mathrm{PRP}}$ is to authenticate a message m for a random key L and to derive the pointer $\mathsf{P} = \mathsf{Point}(\mathsf{K}, \mathsf{L})$ by applying the inverse of a four-round Feistel permutation $\pi^{-1}(\mathsf{K}, \cdot)$ on the ephemeral key L. For the Feistel permutation we use $\mathsf{Mac}(\mathsf{K}, \langle i \rangle_2 \,||\cdot)$ as round function, where $\langle i \rangle_2$ denotes the fixed-length binary representation of the round number $i = 0, 1, 2, 3$ with two bits. To verify a given tuple $(\mathsf{K}, \mathsf{P}, \sigma, m)$ one first recovers L by evaluating the permutation on P and then verifies if (L, σ, m) validates as true. The pseudorandomness of the MAC ensures that the pointer leaks no information about the secret key, nor the ephemeral key.

The construction $\mathsf{DKMAC}_{\mathrm{PRP}}$ is optimal in terms of output length (assuming that keys are uniform bit strings and that at least $|\mathsf{L}|$ additional bits must be communicated for L). Yet, it slightly increases the computational costs, as the

Mac algorithm is now also invoked four times to derive the pointer information (but only on short strings). The construction also shows that neither randomized encryption nor labels are necessary.

For (keyed) pseudorandom round functions f_1, f_2, f_3, f_4 and input $x_0||y_0$ (of equal length parts x_0, y_0), let $x_{i+1}||y_{i+1} = y_i||(x_i \oplus f_i(y_i))$ for $i = 0, 1, 2, 3$. This defines a permutation π (with the round functions and keys given implicitly) mapping input $x_0||y_0$ to output $x_4||y_4$. For our solution here we assume for simplicity that keys L are of even length, such that they can be written as $\mathsf{L} = x_0||y_0$. Instead of using independent round functions we use quasi-independent round functions $f_i = \mathsf{Mac}(\mathsf{K}, \langle i \rangle_2 \,||\cdot)$ by prepending the round number i in binary (represented with the fixed length of two bits).

Construction 1. *Let* $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vf})$ *be a (deterministic) message authentication code. Define* $\mathsf{DKMAC}_{PRP} = (\mathsf{KGen}_{PRP}, (\mathsf{Mac}, \mathsf{Point})_{PRP}, \mathsf{Vf}_{PRP})$ *as follows:*

Key Generation $\mathsf{KGen}_{\mathbf{PRP}}$**.** *The key generation algorithm gets a security parameter* 1^n *and outputs a key* $\mathsf{K} \leftarrow \mathsf{KGen}(1^n)$.

Authentication $(\mathsf{Mac}, \mathsf{Point})_{\mathbf{PRP}}$**.** *The authentication procedure takes as input a secret key* K*, a message* m *and first samples a fresh ephemeral key* $\mathsf{L} \leftarrow \mathsf{KGen}(1^n)$ *by running the key generation of the underlying MAC scheme. For key* L *and input message* m *it computes the tag* $\sigma \leftarrow \mathsf{Mac}(\mathsf{L}, m)$ *and the pointer* $\mathsf{P} \leftarrow \mathsf{Point}(\mathsf{K}, \mathsf{L})$*, where* Point *computes* $\mathsf{P} \leftarrow \pi^{-1}(\mathsf{K}, \mathsf{L})$ *for a four-round Feistel permutation* π *that uses* $\mathsf{Mac}(\mathsf{K}, \langle i \rangle_2 \,||\cdot)$ *as the round functions for* $i = 0, 1, 2, 3$ *and* L *as input. The output of* $(\mathsf{Mac}, \mathsf{Point})_{PRP}$ *is the pair* (σ, P).

Verification $\mathsf{Vf}_{\mathbf{PRP}}$**.** *Upon input a secret key* K*, a pointer* P*, a message* m *and a tag* σ*, it first derives the ephemeral key* $\mathsf{L} = \mathsf{Point}^{-1}(\mathsf{K}, \mathsf{P}) = \pi(\mathsf{K}, \mathsf{P})$ *and outputs* $\mathsf{Vf}(\mathsf{L}, m, \sigma)$.

Correctness of this MAC follows easily form the correctness of the underlying MAC.

Lemma 1. *If* $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vf})$ *is a pseudorandom message authentication code then the delayed-key message authentication scheme* $\mathsf{DKMAC}_{PRP} = (\mathsf{KGen}_{PRP}, (\mathsf{Mac}, \mathsf{Point})_{PRP}, \mathsf{Vf}_{PRP})$ *in Construction 1 is unforgeable against chosen message attacks.*

As for concrete security, the advantage of any adversary $\mathcal{A}_{\mathsf{DKMAC}}$ making q_{MAC} queries of bit length at most l is bounded by q_{MAC} times the advantage of an adversary $\mathcal{A}_{\mathsf{MAC}}$ against the pseudorandomness of MAC that makes $4q_{\mathsf{MAC}}$ queries of length at most $\mathsf{max}(n+2, l)$. Again, the running times of both algorithms are comparable.

Proof. Assume towards contradiction that an adversary $\mathcal{A}$ making q queries $m_1, \ldots, m_q$ to the $\mathcal{O}_{\mathsf{MAC}}(\mathsf{K}, \cdot)$ oracle outputs with non negligible probability a tuple $(\mathsf{P}^*, m^*, \sigma^*)$, s.t. $\mathsf{Vf}^*(\mathsf{K}, \mathsf{P}^*, m^*, \sigma^*)$ but m^* was never submitted to the oracle. Then we can distinguish between two cases:

- $\mathsf{P}^* \neq \mathsf{P}_1, \ldots \mathsf{P}_q$, i.e., the adversary has created a valid forgery for a fresh pointer and thus for a fresh ephemeral key $\mathsf{L}^* \neq \mathsf{L}_1, \ldots \mathsf{L}_q$, since the pointer algorithm is a permutation. Denote the event by E_1.
- $\mathsf{P}^* = \mathsf{P}_i$ for some $i \in \{1, \ldots, q\}$, i.e., the pointer P^* has already appeared in one of the oracle replies. Thus, the adversary $\mathcal{A}$ has successfully forged a MAC for a key L^* after seeing at least one tag $\sigma_i \leftarrow \mathsf{Mac}(\mathsf{L}^*, m_i)$. We denote this event by E_2.

As one of the two cases has to occur if $\mathcal{A}$ is successful —which we denote as the event Win— we have that $\mathrm{Prob}[\mathrm{Win}] \leq \mathrm{Prob}[E_1] + \mathrm{Prob}[E_2]$ (note that events E_1, E_2 both require a success). We show in the full paper that in both cases we can construct an adversary that breaks the underlying MAC scheme. □

Lemma 2. *The delayed-key MAC scheme* DKMAC_{PRP} *in Construction 1 is leakage-invariant.*

Proof. To prove leakage-invariance we have to show that for every adversary $\mathcal{A}$ with oracle access to $\mathcal{O}_{\mathsf{MAC}}(\mathsf{K}, \cdot)$ and $\mathsf{Vf}(\mathsf{K}, \cdots)$ that predicts with noticeable probability some information $f(\mathsf{K})$ about the key K, we can derive an adversary $\mathcal{B}$ that only has access to $\mathsf{Mac}(\mathsf{K}, \cdot)$ and $\mathsf{Vf}(\mathsf{K}, \cdot)$ but predicts $f(\mathsf{K})$ with the same advantage as $\mathcal{A}$.

Assume that $\mathcal{A}$ is able to derive some non-trivial information about K after sending q queries to its $\mathcal{O}_{\mathsf{MAC}}$ and Vf oracles, which implements the authentication process of our delayed-key MAC. Then we can construct an adversary $\mathcal{B}$ that successfully determines $f(\mathsf{K})$ when sending $4q$ queries to its $\mathsf{Mac}(\mathsf{K}, \cdot)$ and $\mathsf{Vf}(\mathsf{K}, \cdots)$ oracles. To this end, $\mathcal{B}$ mimics the $\mathcal{O}_{\mathsf{MAC}}$ oracle by computing the tag $\sigma_i \leftarrow \mathsf{Mac}(\mathsf{L}_i, m_i)$ for any query m_i and some self-chosen key L_i and calculating P_i with the help of its own oracle (and analogously for verification requests). Thus, for each of $\mathcal{A}$'s queries, $\mathcal{B}$ has to invoke $\mathsf{Mac}(\mathsf{K}, \cdot)$ four times to simulate $\mathcal{O}_{\mathsf{MAC}}$ or Vf. If $\mathcal{A}$ outputs some information a, $\mathcal{B}$ forwards it as its own output. Since the simulation is perfect from $\mathcal{A}$'s point of view the success probabilities of $\mathcal{B}$ and $\mathcal{A}$ are identical. □

The construction $\mathsf{DKMAC}_{\mathrm{PRP}}$ is already optimal concerning the communication overhead (assuming, that at least $|\mathsf{L}|$ additional bits have to be communicated) but increases the computational costs by four additional evaluations of the underlying MAC. Our second construction of an unbounded delayed-key MAC, which we discuss in detail in the full version, requires less Mac computations (two instead of four) but comes with larger output lengths.

5 One-Sided Delayed-Key MACs: The Bounded Case

In this section we show that, by reducing the security requirements for unforgeability and leakage-invariance, we can construct key-delayed MACs that require less Mac invocations than our previous constructions or are even optimal in both, computational costs and output length. In other words, we can trade in security for efficiency. First, we bound the adversaries against unforgeability and

leakage-invariance to make at most $O(\log(n))$ many verification queries, which allows to obtain a construction that requires only two MAC computations and is almost optimal in terms of output length. We present the construction in the full version of the paper, where we also show that the scheme is even strongly leakage-invariant (meaning that $\mathcal{B}$ does not make more queries than $\mathcal{A}$), as long as we only demand that $\mathcal{A}$ is unable to predict the entire key.

By further restricting the adversary against the leakage-invariance to make only a single authentication query, we obtain our most efficient solution that requires no additional Mac computations and has optimal output length . Note that the underlying MAC is then assumed to be secure against related-key attacks.

As already mentioned in the introduction, limiting the number of verification queries corresponds to the common approach that in key-exchange protocols, both server and client verify only a single MAC each. Leakage-invariance for only $\mathcal{F} = \{\mathrm{ID}\}$ is sufficient, if the key gets afterwards hashed by a hash function that behaves like a random oracle.

5.1 XOR-Construction

In our most simple and efficient construction, we use the shared key K to directly mask the ephemeral key. That is, by computing the one-time-pad encryption of L under K, i.e., $\mathsf{P} = \mathsf{K} \oplus \mathsf{L}$. Thus, for any authentication query, $\mathsf{DKMAC}_\oplus$ makes only a single Mac computation.

Definition 7. *Let* $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vf})$ *be a message authentication code. Define the delayed-key* $\mathsf{DKMAC}_\oplus = (\mathsf{KGen}_\oplus, (\mathsf{Mac}, \mathsf{Point})_\oplus, \mathsf{Vf}_\oplus)$ *as follows*

Key Generation $\mathsf{KGen}_\oplus$**.** *The key generation algorithm gets a security parameter* 1^n *and outputs a key* $\mathsf{K} \leftarrow \mathsf{KGen}(1^n)$.

Authentication $(\mathsf{Mac}, \mathsf{Point})_\oplus$**.** *The authentication procedure takes as input a shared secret key* K*, a message* m *and outputs* $\sigma \leftarrow \mathsf{Mac}(\mathsf{L}, m)$ *and pointer* $\mathsf{P} = \mathsf{K} \oplus \mathsf{L}$ *for a randomly chosen* $\mathsf{L} \leftarrow \mathsf{KGen}(1^n)$.

Verification $\mathsf{Vf}_\oplus$**.** *Upon input a secret key* K*, a pointer* P*, a message* m *and a tag* σ *it outputs* $\mathsf{Vf}(\mathsf{P} \oplus \mathsf{K}, m, \sigma)$.

Correctness of $\mathsf{DKMAC}_\oplus$ *follows from the correctness of the underlying MAC.*

In order to prove the unforgeability of our $\mathsf{DKMAC}_\oplus$ construction, we require a stronger assumption on the underlying MAC, namely that it is a related-key secure pseudorandom function. The first formal security model for related key attacks was introduced by Bellare and Kohno in [3]. Inter alia, they have shown that PRFs that are provably secure against those attacks can be achieved when the set of relations is restricted to some non-trivial class of key transformation functions, denoted by Φ. The notion for Φ-related-key security then extends the notion of standard PRF's and grants the adversary access to a related-key oracle that is either $\mathsf{Mac}_{\mathrm{RK}(\cdot,k)}(\cdot)$ or $f_{\mathrm{RK}(\cdot,k)}(\cdot)$. In both cases a key k is chosen at random and in the random world, also a function f gets chosen randomly. Each query of the adversary then consists of a key transformation function $\phi : \mathcal{K} \to \mathcal{K}$

and an input value m. The query is answered by $\mathsf{Mac}(\phi(k), m)$ and $f(\phi(k), m)$ respectively.

Definition 8. *Let Φ be a set of key transformation functions, and $\mathcal{D}$ an adversary with access to related-key oracles that is allowed to send queries $(\phi, m) \leftarrow \Phi \times \mathcal{M}$. A pseudorandom* Mac *is called* secure against related-key attacks *if for any efficient algorithm $\mathcal{D}$ the advantage*

$$\left|\text{Prob}\left[\mathcal{D}^{\mathsf{Mac}_{RK(\cdot,k)}(\cdot)}(1^n) = 1\right] - \text{Prob}\left[\mathcal{D}^{f_{RK(\cdot,k)}(\cdot)}(1^n) = 1\right]\right|$$

is negligible, where the probability in the first case is over $\mathcal{D}$'s coin tosses and the choice of $k \leftarrow \mathsf{KGen}(1^n)$, and in the second case over $\mathcal{D}$'s coin tosses, the choice of the random function $f : \mathcal{K}_n \times \mathcal{M}_n \to \mathcal{R}_n$ and random $k \leftarrow \mathcal{K}_n$.

Note that related-key secure pseudorandom MACs are unforgeable with respect to related-key attacks, too.

For our construction we need related-key security only for one class of transformations, that is the function that adds a given value $\Delta \in \{0,1\}^n$ to the hidden key K. Sticking to the notation of [3] we denote this function by $\text{XOR}_\Delta : \mathcal{K} \to \mathcal{K}$ and the resulting class of functions by $\Phi_n^{\oplus} = \{\text{XOR}_\Delta : \Delta \in \{0,1\}^n\}$. Constructions for $\Phi_n^{\oplus}$-related-key secure pseudorandom functions were proposed in [15].

Lemma 3. *If $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vf})$ is a pseudorandom message authentication code secure against related-key attacks for the relation $\Phi_n^{\oplus}$, then the delayed-key MAC scheme $\mathsf{DKMAC}_\oplus = (\mathsf{KGen}_\oplus, (\mathsf{Mac}, \mathsf{Point})_\oplus, \mathsf{Vf}_\oplus)$ in Construction 7 is unforgeable against chosen message attacks, if the adversary makes at most $O(\log(n))$ verification queries.*

A closer look at the concrete security reveals that the advantage of any adversary $\mathcal{A}_{\mathsf{DKMAC}}$ making $q_{\mathsf{MAC}}, q_{\mathsf{Vf}}$ queries each of length at most l, is bounded by $2^{q_{\mathsf{Vf}}}$ times the advantage of an adversary $\mathcal{A}_{\mathsf{MAC}}$ against the related-key pseudorandomness of MAC that makes q_{MAC} queries of length at most l.

Proof. Assume towards contradiction that an adversary $\mathcal{A}$ after learning several tags $(\sigma_1, \mathsf{P}_1), \ldots, (\sigma_, \mathsf{P}_q)$ from its oracle $\mathcal{O}_{\mathsf{MAC}}(\mathsf{K}, \cdot)$ is able to compute a forgery $(\mathsf{P}^*, m^*, \sigma^*)$ with $m^* \neq m_1 \ldots m_q$. Then we can construct an adversary $\mathcal{A}_{\mathsf{MAC}}$ breaking the related-key unforgeability of the underlying MAC.

Our adversary $\mathcal{A}_{\mathsf{MAC}}$ has black-box access to a related-key oracle $\mathsf{Mac}_{\text{RK}(\cdot,\mathsf{L})}(\cdot)$ and uses $\mathcal{A}$ to produce a forgery $(\Delta^*, m^*, \sigma^*)$ for some key $\mathsf{L} \oplus \Delta^*$. For the sake of readability it is assumed, that the real key transformation XOR is already included in the oracle and the adversary has only to provide some value $\Delta \in \{0,1\}^n$.

When $\mathcal{A}$ sends the first authentication query m_1, $\mathcal{A}_{\mathsf{MAC}}$ invokes its own oracle on $(0^n, m_1)$ receiving $\sigma_1 = \mathsf{Mac}(\mathsf{L}, m_1)$ which he passes together with a randomly chosen P back to $\mathcal{A}$. The value P can also be seen as $\mathsf{L} \oplus \mathsf{K}$ for some unknown K. Due to the pseudorandomness of Mac, the tag σ_1 does not leak any information about the applied key L. Thus, from $\mathcal{A}$'s point of view the value P is

indistinguishable from a real one-time-pad encryption of some secret key K. For any further authentication query m_i of $\mathcal{A}$, our adversary chooses a random Δ_i and sends (Δ_i, m_i) to its own oracle. The adversary $\mathcal{A}_{\mathsf{MAC}}$ then responds with the answer σ_i and a pointer $\mathsf{P}_i = \mathsf{P} \oplus \Delta_i$.

When $\mathcal{A}$ wants to query its verification oracle, our adversary $\mathcal{A}_{\mathsf{MAC}}$ has to guess the answer bit, otherwise it might send the message of the potential forgery to his tagging oracle, thereby nullifying the message for its own output. Thus, whenever $\mathcal{A}$ makes a verification query, $\mathcal{A}_{\mathsf{MAC}}$ halts $\mathcal{A}$ and then runs two instantiations for the answer bit $b = 0$, resp. $b = 1$. Hence, for efficiency reasons we allow $\mathcal{A}$ to make at most $O(\log(n))$ queries to the verification oracle.

If, at the end, each of the at most n instantiations of $\mathcal{A}$ holds with a forgery $(\mathsf{P}^*_j, m^*_j, \sigma^*_j)$, our adversary $\mathcal{A}_{\mathsf{MAC}}$ guesses an index $j \in \{1, \ldots, n\}$. It then computes $\Delta^* = \mathsf{P}^*_j \oplus \mathsf{P}$ and outputs $(\Delta^*, m^*_j, \sigma^*_j)$ as its own forgery. Overall, $\mathcal{A}_{\mathsf{MAC}}$ succeeds with probability $1/\mathrm{poly}(n)$ times the success probability of $\mathcal{A}$, which contradicts the assumption that MAC is related-key unforgeable. □

Lemma 4. *The delayed-key MAC scheme* $\mathsf{DKMAC}_{\oplus}$ *in Construction 7 is* $(1, O(\log(n)), \{ID\})$*-leakage invariant.*

Proof. If there exists an adversary $\mathcal{A}$ that outputs with non-negligible probability the complete secret key K after it received a tag $(\sigma, \mathsf{P}) \leftarrow \langle \mathsf{Mac}(\mathsf{L}, m), \mathsf{K} \oplus \mathsf{L} \rangle$ for some random L and chosen m, we can derive an adversary $\mathcal{B}$ that is able to extract K only from $\sigma \leftarrow \mathsf{Mac}(\mathsf{K}, m)$ for some chosen m as well.

The idea is that by determining K, also the key L can be obtained unambiguously. Thus, when we construct the adversary $\mathcal{B}$ that uses $\mathcal{A}$, its target key K actually plays the role of L in the game of $\mathcal{A}$. Thus, when $\mathcal{B}$ receives the authentication query m from $\mathcal{A}$ it triggers its oracle $\mathsf{Mac}(\mathsf{K}, \cdot)$ on m and passes the answer σ together with a randomly chosen pointer P back to $\mathcal{A}$. The pointer value then corresponds to the one-time-pad encryption of K with some random, secret key L.

For any verification query $(\mathsf{P}_i, m_i, \sigma_i)$ of $\mathcal{A}$, the adversary $\mathcal{B}$ first checks whether $\mathsf{P}_i = \mathsf{P}$. If so, it forwards the query to its $\mathsf{Vf}(\mathsf{K}, \cdot)$ oracle, otherwise it has to "guess" the answer bit. To this end, $\mathcal{B}$ runs two instantiations of $\mathcal{A}$, for each $b = 0, 1$. Since we allow $\mathcal{A}$ to make only at most $O(\log(n))$ verification queries, $\mathcal{B}$ starts at most n instantiations.

Finally, each instantiation of $\mathcal{A}$ stops, outputting its guess a_j that corresponds to some L_j in $\mathcal{B}$'s game. To determine the right key, adversary $\mathcal{B}$ computes for each $j = 1, 2, \ldots, n$ the potential counterpart $\mathsf{K}_j = \mathsf{P} \oplus \mathsf{L}_j$ and outputs K_j where $\sigma = \mathsf{Mac}(\mathsf{K}_j, m)$.

Due to the limitation of a single authentication query, our adversary $\mathcal{B}$ is able to simulate the oracle $\mathcal{O}_{\mathsf{MAC}}$ of $\mathcal{A}$ perfectly, such that $\mathcal{B}$ succeeds with the same probability as $\mathcal{A}$. □

6 Two-Sided Delayed-Key MACs: A Feasibility Result

In this section we discuss that two-sided delayed-key MACs are realizable without relying on collision-resistance. The idea —explained in the setting of key

exchange— is to use a signature scheme to authenticate each transmitted message immediately (such that both parties basically only have to store keys for the MAC), and to finally MAC the public key of the signature scheme.

Note that the existence of one-way functions is shown to be necessary and sufficient for the existence of secure signature schemes in [18]. As we, in addition, only require unforgeability from the underlying MAC, the security of our construction formally relies only on one-way functions. Yet, applying a signature scheme for each message is very expensive, of course. Hence, this construction should be seen as a feasibility result only. We leave it as an interesting open problem to find an efficient construction for this scenario.

Note that in order to turn the idea above into a formal solution we need to change the notion of unforgeability and leakage-invariant slightly. Namely, we assume that the adversary $\mathcal{A}$ in both cases now can pass another parameter `keep` or `pointer` (besides m_i, ℓ_i) to oracle $\mathcal{O}_{\mathsf{MAC}}$. For parameter `keep` the oracle returns tags σ_i for the previously selected ephemeral key K and only if queried for `pointer` it returns the pointer P and generates a new ephemeral key. An adversary $\mathcal{A}$ against the unforgeability is then deemed successful if it outputs a tuple $(\mathsf{P}^*, \bar{m}^*, \bar{\ell}^*, \sigma^*)$ with $\mathsf{Vf}(\mathsf{K}, \mathsf{P}^*, \bar{m}^*, \bar{\ell}^*, \sigma^*) = 1$ and $\mathcal{A}$ has never issued $(\bar{m}^*, \bar{\ell}^*) = ((m_1{}^*, \ell_1{}^*), \ldots, (m_n{}^*, \ell_n{}^*))$ between two `pointer` queries to $\mathcal{O}_{\mathsf{MAC}}(\mathsf{K}, \cdot)$.

The $\mathsf{DKMAC}_{\mathsf{two}}$ *Construction.* Recall the notion of signature schemes: a signature scheme consists of three efficient algorithms $(\mathsf{SKGen}, \mathsf{SSign}, \mathsf{SVf})$ where SKGen on input 1^n returns a key pair (sk, pk); algorithm SSign on input sk and a

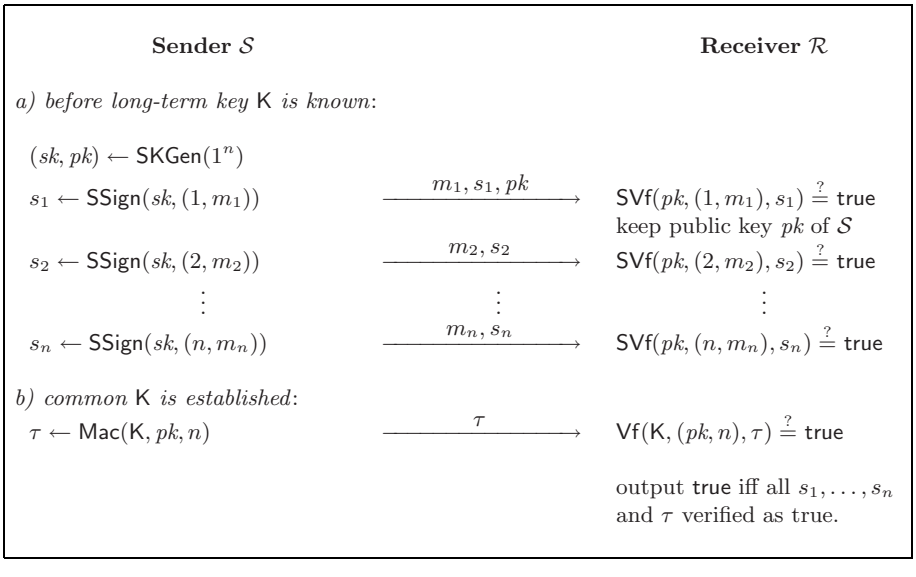

Fig. 1. $\mathsf{DKMAC}_{\mathsf{two}}$: Two-sided Delayed-Key MAC

message $m \in \{0,1\}^*$ returns a signature s; and algorithm SVf for input pk, m, s returns a decision bit. We assume completeness in the sense that any signature generated via SSign is also accepted by SVf. *Unforgeability* of signature schemes is defined analogously to unforgeability of MACs, but now the adversary gets as input the public key pk instead of the security parameter 1^n and has access to a signing oracle $\mathsf{SSign}(sk, \cdot)$.

Our construction $\mathsf{DKMAC}_{\mathsf{two}}$ (incorporated into a key exchange protocol) is given in Figure 1. Note that the sender only needs to store the key pair (sk, pk) and the receiver merely stores pk and a bit indicating any error in the verifications so far. Formally, we can let $\mathsf{Mac}(\mathsf{L}, m, \ell)$ be the algorithm which for $\mathsf{L} = (sk, pk) \leftarrow \mathsf{SKGen}(1^n)$ outputs $\sigma = (pk, \mathsf{SSign}(sk, m, \ell))$. The point algorithm $\mathsf{Point}(\mathsf{K}, \mathsf{L}, \ell)$ returns a MAC value P of pk under key K for an unforgeable MAC. Then an adversary against the key exchange protocol can be easily cast in our extended unforgeability and leakage-invariance model. This adversary calls $\mathcal{O}_{\mathsf{MAC}}$ several times with (i, m_i, ℓ_i) for parameter `keep` and subsequently eventually calls the oracle about parameter `pointer` to retrieve the MAC of the public key under K.

Unforgeability and Leakage-Invariance of $\mathsf{DKMAC}_{\mathsf{two}}$. The $\mathsf{DKMAC}_{\mathsf{two}}$ construction is unforgeable if the underlying signatures scheme is unforgeable against chosen-message attacks and the underlying MAC is unforgeable as well. The unforgeability of the MAC and the fact that collisions among independently generated keys are unlikely implies that the adversary can only use a previously chosen public key by $\mathcal{O}_{\mathsf{MAC}}$ (or else forges a MAC under K for a new key pk^*). But then the adversary must forge a signature for a tuple (i^*, m^*, ℓ^*) which has not been signed before under this public key. By the unforgeability of the signature scheme this cannot happen with more than negligible probability.

Obviously, the scheme $\mathsf{DKMAC}_{\mathsf{two}}$ is strongly leakage-invariant, as it uses the secret long-term key K only for a single computation of the underlying MAC.

Online Verification with Immediate Abort. In the context of online verification it might be desirable that the verifier can abort the authentication process as soon as he receives the first invalid tag. To this end, we augment the usual verification algorithm Vf of DKMAC's such that it allows online processing: $\mathsf{Vf}'(\mathsf{K}, \mathsf{P}, m, \ell, \sigma, \mathsf{st})$ now also expects some state information st which can either be `keep` or `pointer`. On input `keep` the algorithm Vf' returns $\mathsf{Vf}(m, \ell, \sigma)$ and for `pointer` it outputs $\mathsf{Vf}(\mathsf{K}, \mathsf{P}, m, \ell, \sigma)$. Thus, as long as the long-term key K is unknown, the verifier runs $\mathsf{Vf}'(\perp, \perp, m_i, \ell_i, \sigma_i, \texttt{keep})$ and aborts when it receives 0, indicating an invalid tag. Obviously, our construction $\mathsf{DKMAC}_{\mathsf{two}}$ allows for online verification with immediate abort as the verifier can check, while being in `keep`-mode, if $\mathsf{SVf}(pk, (i, m_i), s_i) = \mathsf{true}$ and abort the authentication as soon as the first verification fails.

Acknowledgments

We thank Yevgeniy Dodis, Stefan Lucks and the anonymous reviewers for valuable comments. Both authors are supported by the Emmy Noether Program Fi 940/2-1 of the German Research Foundation (DFG).

References

1. Bellare, M.: New Proofs for NMAC and HMAC: Security without Collision- Resistance. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 602–619. Springer, Heidelberg (2006)
2. Bellare, M., Goldreich, O.: A. Mityagin The Power of Verification Queries in Message Authentication and Authenticated Encryption. Number 2004/309 in Cryptology eprint archive (2004), eprint.iacr.org
3. Bellare, M., Kohno, T.: A Theoretical Treatment of Related-Key Attacks: RKA-PRPs, RKA-PRFs, and Applications. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 491–506. Springer, Heidelberg (2003)
4. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated Key Exchange Secure against Dictionary Attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (2000)
5. Advanced Security Mechanism for Machine Readable Travel Documents Extended Access Control (EAC). Technical Report (BSI-TR-03110) Version 2.0 Release Candidate, Bundesamt fuer Sicherheit in der Informationstechnik, BSI (2008)
6. Canetti, R.: Universally Composable Security: A new Paradigm for Cryptographic Protocols. In: Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS) 2001. IEEE Computer Society Press, Los Alamitos (2001), for an updated version see: eprint.iacr.org
7. Fischlin, M.: Security of NMAC and HMAC Based on Non-malleability. In: Malkin, T.G. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 138–154. Springer, Heidelberg (2008)
8. Gennaro, R.: Faster and Shorter Password-Authenticated Key Exchange. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 589–606. Springer, Heidelberg (2008)
9. Garay, J.A., Kolesnikov, V., McLellan, R.: MAC Precomputation with Applications to Secure Memory. In: Samarati, P., et al. (eds.) ISC 2009. LNCS, vol. 5735, pp. 427–442. Springer, Heidelberg (2009)
10. Goldwasser, S., Micali, S.: Probabilistic Encryption. Journal of Computer and System Science 28(2), 270–299 (1984)
11. Gennaro, R., Rohatgi, P.: How to Sign Digital Streams. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 180–197. Springer, Heidelberg (1997)
12. Halevi, S., Krawczyk, H.: Strengthening Digital Signatures Via Randomized Hashing. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 41–59. Springer, Heidelberg (2006)
13. Jablon, D.: Strong password-only authenticated key exchange. ACM Computer Communications Review 26(5), 5–26 (1996)
14. Katz, J., Ostrovsky, R., Yung, M.: Efficient Password- Authenticated Key Exchange Using Human-Memorable Passwords. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, p. 475. Springer, Heidelberg (2001)

15. Lucks, S.: Ciphers Secure against Related-Key Attacks. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 359–370. Springer, Heidelberg (2004)
16. Perrig, A., Canetti, R., Song, D., Tygar, J.D.: The TESLA Broadcast Authentication Protocol. In: CryptoBytes, vol. 5, pp. 2–13. RSA Security (2002)
17. Rescorla, E.: SSL and TLS: designing and building secure systems. Addison-Wesley, Reading (2001)
18. Rompel, J.: One-Way Functions are Necessary and Sufficient for Secure Signatures. In: Proceedings of the Annual Symposium on the Theory of Computing (STOC) 1990, pp. 387–394. ACM Press, New York (1990)