A Leakage Resilient MAC

Dan Martin, Elisabeth Oswald, and Martijn Stam

Department of Computer Science, University of Bristol,
Merchant Venturers Building, Woodland Road,
Bristol, BS8 1UB, United Kingdom.
{dan.martin, elisabeth.oswald, martijn.stam}@bris.ac.uk

Abstract. We put forward a message authentication code (MAC) for which we claim a high degree of resilience against a key-recovering attacker expoiting practical side channels. We achieve this by blending the lessons learned from many years of engineering with the scientific approach provided by leakage resilience. This highlights how the two often disparate fields can benefit from each other.
Our MAC is relatively simple and intuitive: we essentially base our construction on bilinear groups and secret share out our key. The shares are then refreshed before each time they are used and the algebraic properties of the bilinear pairing are used to compute the tag without the need to reconstruct the key. This approach allows us to prove (in the random oracle model) existential unforgability of the MAC under chosen message attacks in the presence of (continuous) leakage, based on two novel assumptions: a bilinear Diffie–Hellman variant and an assumption related to how leaky performing a group operation is.
In practice we envision our scheme would be implemented using pairings on some pairing friendly elliptic curve, where the leakiness of the group operation can be experimentally estimated. This allows us to argue about practical implementation aspects and security considerations of our scheme. We compare our scheme against other leakage resilient MACs (or related schemes) that have appeared in the literature and conclude ours is both the most efficient and by far the most practical.

1 Introduction

Current research into cryptographic schemes resilient to side channel attacks takes place in two almost entirely separated 'camps'. On the one hand there is the 'engineering' community that is primarily focused on practice and whose long line of research can be regarded as a cat and mouse game—a countermeasure is proposed, implemented, and evaluated against a very specific (set of) attack(s). Later the same countermeasure is picked up again, some of the assumptions made by the proposer are re-evaluated and then the countermeasure is broken, probably a fix is reported, etc. On the other hand there is the 'scientific' community. To them the engineering approach of break-and-patch can seem rather ad-hoc and lacking in rigour. Instead they advocate the by now well established field of leakage resilient cryptography. Here the primary goal of researchers is the design of cryptographic constructions that, per design, can provably tolerate some leakage on the key, or state, per invocation (aka the continuous leakage model). However, from an engineering perspective, this theoretical approach can seem overblown and irrelevant: the resulting constructions tend to be high in overhead, and the leakage is modelled in a way that is both too powerful (viz. leakage is expressed as any efficiently computable function on the input, bearing no relation to what is actually being computed) and too weak (viz. the leakage is local and can be captured by some relatively small bitstring).

Only a few works have been proposed that attempt to find some middle-ground: schemes that come with some provable guarantee against arbitrary leaks without incurring prohibitively high overhead, which can be implemented securely also against state of the art side channel attacks. For instance, very recently Faust et al. [8] showed that when relaxing security notions from completely adaptive inputs (i.e. adversaries may choose input messages but also the side channel leaks adaptively) to non-adaptive security, simpler constructions for symmetric key cryptography can be achieved than previously thought. In a differently

motivated publication, Balasch et al. [1] take a provably secure method, i.e. inner product masking, trim it down to implement a masked AES with it, and show this leads to a result which is comparable to other state of the art, yet not formally proven, masking approaches.

One notable gap in the literature on leakage resilient cryptographic schemes is with respect to message authentication codes (MACs). Despite the need for authentication at the start of almost any cryptographic interaction, and hence its wide deployment in practice, there exists only one leakage resilient MAC construction in the literature [12]. We propose a new MAC scheme that is motivated by engineering constraints, yet at the same time benefits from a rigorous justification using the framework of leakage resilience.

Common ground. Before discussing our own proposal in more detail, we want to run through some key ideas that underpin it. These core principals have been independently recognised as useful by both practitioners and theorists.

The first key idea is that of frequent 're-keying': the simple idea here is that one can thwart DPA [14] style attacks (i.e. attacks that require several side channel observations with known input data on one secret key) by implementing a scheme that always refreshes its keying material after a single (or very few) uses of the secret key. Leakage resilient cryptography fundamentally depends on this idea, but also in practical contributions this idea has been put forward in several, slightly different ways. The first proposal (by Kocher [13]) is to supply a device with a 'clue' about how to update the key, e.g. a random number for each message with which one traverses through a binary hash tree that updates the key. This scheme is stateful, i.e. it necessitates some form of synchronisation between parties. A second option is to generate such a secret internally (leak free), use it to update the key via a simple operation, and then send it to the receiver together with the output of the cryptographic operation [16]. In the next invocation the same original key is used for updating. This method is hence not stateful. The security argument that practitioners use is as follows: assuming the key update is secure against SPA and DPA, and the cryptographic operation is secure against SPA, the overall system is secure. For neither of these two constructions a formal security proof is provided, and given the nature of the constructions it also seems unlikely that a formal proof can be achieved.

The second key idea used in our work relates to the technique of secret sharing a key. Secret sharing is well researched from a practical perspective: under the names of 'masking' and 'higher order masking' known implementation techniques exist and are widely deployed especially in the context of symmetric cryptography. The same principle has also been applied from a theoretical perspective to public key schemes; for instance, Kiltz and Pietrzak [15] use this technique in their construction of a leakage resilient public key encryption scheme.

The third key idea combines the first two: once the key is split in shares, it is possible to refresh the shares of the key by using some internal randomness. Thus, while the *key* stays the same, its *representation* is updated locally, which gives the same benefits (described above) as rekeying. Kiltz and Pietrzak [15] used this approach, where they identified groups where the resharing does and does not seem to work. Importantly, they realise that instantiating the secret sharing over a group with a hard discrete logarithm problem connects practice with theory and allows some provable guarantees for practical secret sharing. Eventually they settled for using elliptic curves in a pairing setting: this allowed them to perform the public key operations without the need to ever reconstruct the key. The cryptographic scheme is effectively executed twice and the recombination of the shares (of the desired outcome) is a simple operation. Where Kiltz and Pietrzak constructed a leakage-resilient version of ElGamal encryption, Galindo and Vivek [10] used the same setting to create a leakage resilient signature scheme. These constructions lend themselves to a security proof based on the 'only computation leaks information' (OCLI) paradigm [17]. However it

also enables a practical implementation with manageable overhead: sharing out the key implies that (in practice) we can aim for a timely separation between computations on these shares, which gives some practical credibility to using OCLI in a proof. The disadvantage of both the Kiltz–Pietrzak and the Galindo–Vivek scheme is their explicit dependence on the generic group model, rather than on a clean, falsifiable assumption.

Our contribution. Inspired by the bilinear ElGamal cryptosystem by Kiltz and Pietrzak [15], we propose a MAC scheme that combines the three simple key ideas described above (frequent re-keying, use of secret sharing, update of shares as re-keying mechanism) in a setting with pairings. As a result, our scheme can be proven secure against side channel key recovery attacks and against existential unforgery in the presence of continuous side channel leakage. However, in contrast to the earlier works [15, 10] we are able to prove our result *without* reference to the generic group model.

In Sect. 2 we set the stage and prove our scheme secure, in the usual sense, based on a bilinear assumption and in the random oracle model. Next, in Sect. 3 we discuss the key update procedure in more detail and define what it means for the key update scheme to be secure against leakage, giving both a computational and decisional variant of the relevant security game. This allows us in Sect. 4 to combine the MAC and the key update procedure that is i) leakage resilient against key recovery if the key updates are computationally secure, and ii) leakage resilient against existential forgeries if the key update are decisionally secure. We discuss implementation requirements based on proof assumptions, and compare our construction to other work in Sect. 5.

Our approach introduces a useful modular view that bridges the practical and theoretical. On the one hand, all the assumptions hold in the generic group model, leading to an efficient, leakage resilient MAC that can tolerate to leak slightly less than half the bits per invocation. On the other hand, in a practical setting our scheme remains secure and it is possible to test the assumptions we make on the leakage resilience of the key update function independently. As said before, in practice rekeying can be used to thwart DPA, but SPA security still needs to be provided. For our MAC the same holds true, namely that the operations used for the MAC still need to be resistant against SPA. However, because we have proven our scheme leakage resilient, the *level* of SPA resistance is noticeably lower: even if we leak a fair number of bits, security will be maintained (compare this to an implementation of ElGamal signatures, where leakage of only a few bits of the nonces renders the entire scheme insecure due to lattice attacks).

Overall, the result is rather fascinating: we end up with a leakage resilient MAC (a symmetric primitive) using pairing based cryptography! Because we spell out the key update as a separate entity in both proof and construction we can deliver a 'modular' scheme. Consequently, the security of the key update becomes better understood (either in theory or practice), our MAC construction will automatically benefit. Moreover, and we believe this to be the main attraction of our proposal, 'despite' being able to prove the scheme secure in the presence of leakage, our construction remains simple, intuitive, and relevant.

2 A Bilinear MAC

Our MAC scheme is based on bilinear groups. Before spelling it out, we introduce relevant notation and the underlying hard problems that underlie its security.

If A is a deterministic algorithm we write $y \leftarrow A(x)$ to denote that A outputs y on input x. If A is probabilistic we will write $y \xleftarrow{\$} A(x)$ or $y \xleftarrow{r} A(x)$ if we need to use the randomness r explicitly later. Unless specified differently, adversarial algorithms will are probabilistic.[1] We use $\mathcal{K}$ to represent a

[1] Since we stick to concrete reductions, we do not need to specify the complexity class to which adversaries belong.

distribution, $\mathbb{G}$ to represent groups and $[q]$ to represent the set of elements $1 \dots q$. In each case we write $k \xleftarrow{\$} \mathcal{K}$, $g \xleftarrow{\$} \mathbb{G}$ and $i \xleftarrow{\$} [q]$ to denote selecting an element uniformly at random.

We briefly recall the definition of bilinear groups and of bilinear maps, where we adhere to asymmetric pairings (see e.g. Galbraith et al. [9] for an overview). Let $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ be cyclic groups all of prime order q with generators g, h, and f, respectively. A bilinear map is a function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the following properties:

- $\forall u \in \mathbb{G}_1, v \in \mathbb{G}_2, a, b \in \mathbb{Z}_q : \ e(u^a, v^b) = e(u, v)^{ab}$;
- $e(g, h) \neq 1$ (this is called non-degeneracy).

2.1 Yet Another Diffie–Hellman Problem

In Definition 1 we introduce a bilinear problem, which we coin the target bilinear Diffie–Hellman (TBDH) problem. While we could not find prior mention of the TBDH problem in the literature, it is relatively straightforward to relate it to known problems. In Theorem 1 we provide a simple reduction: we show that if an adversary A can break TBDH, then we can construct a related adversary B (that runs A) who can break the co-bilinear Diffie–Hellman (CBDH) problem. If CBDH is assumed to be a hard problem,[2] no such efficient B can exist and, by consequence no efficient and successful A can exist. This implies that if the CBDH problem is hard, then so is the TBDH problem. Similarly, it can be shown that if the standard computational Diffie–Hellman (CDH) Problem is easy in $\mathbb{G}_T$ then the TBDH problem is easy. This means that the TBDH problem's difficulty is sandwiched between the hardness of CDH and CBDH, both of which are well studied hardness assumptions (and believed hard for the usual pairing groups based on elliptic curves).

Definition 1 (Target Bilinear Diffie–Hellman Problem). *Let $\mathbb{G}_1 = \langle g \rangle, \mathbb{G}_2 = \langle h \rangle, \mathbb{G}_T$ be three groups of order q with the non-degenerate bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ between them. We say the Target Bilinear Diffie–Hellman (TBDH) Problem is hard if given h^x, f^y it is hard to compute f^{xy}, where x, y are sampled uniformly at random from $\mathbb{Z}_q$. Given an adversary A we define their advantage of winning this game as $\mathbf{Adv}^{tbdh}(A) = \Pr\left[\mathcal{A} = f^{xy} : \mathcal{A} \leftarrow A(g, h, h^x, f^y)\right]$.*

Theorem 1. *Let A be an adversary against the Target Bilinear Diffie-Hellman Problem, then there exists an adversary B (with approximately the same runtime as A) against the Co-Bilinear Diffie-Hellman problem (CBDH) [22] (given $h, h^a, h^b \in \mathbb{G}_2$ and $g \in \mathbb{G}_1$, find f^{ab}), such that:*

$$\mathbf{Adv}^{tbdh}(A) \leq \mathbf{Adv}^{cbdh}(B) \ .$$

Proof. Let adversary A against TBDH be given, then adversary B that breaks CBDH is given as follows.

adversary $B(g, h, h^a, h^b)$:
$f^a \leftarrow e(g, h^a)$
$e(g, h)^{ab} \leftarrow A(g, h, h^b, f^a)$
Return $e(g, h)^{ab}$

From this it is straightforward to check that $\mathbf{Adv}^{tbdh}(A) \leq \mathbf{Adv}^{cbdh}(B)$. □

[2] Where we deliberately leave the notion of hardness informal; of course it is possible to modify our results to the usual notions of negligible advantages against probabilistic polynomial-time adversaries.

proc $KG()$:
$K \xleftarrow{\$} \mathbb{G}_1$
Return K

proc $TAG(K, m)$:
$w \xleftarrow{\$} R$
$W \leftarrow H(m, w)$
$T \leftarrow e(K, W)$
Return (T, w)

proc $VRFY(K, (T, w), m)$:
$W \leftarrow H(m, w)$
$T' \leftarrow e(K, W)$
Return $(T' = T)$

Fig. 1. Our bilinear MAC scheme $\mathcal{M}$

2.2 The Bilinear MAC Construction

Let $\mathbb{G}_1 = \langle g \rangle, \mathbb{G}_2 = \langle h \rangle$ be groups of order q and let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a non-degenerate bilinear pairing as before. Let $H : M \times R \to \mathbb{G}_2$ be a hash function with message space M and randomness space R. Using these building blocks, our MAC scheme $\mathcal{M} = (KG, TAG, VRFY)$ is defined in Fig. 1. Key generation consists of generating a random group element. Tag generation first hashes a message with a random value, then takes the resulting hash as input to a bilinear map (using the secret key as other input). A MAC consists hence of a message, its tag, and the internal randomness. Verification simply reconstructs the tag T.

We can prove our MAC secure in the usual context, i.e. we can show that it provides existential unforgeability in the context of chosen message attacks (EUF-CMA). Thm. 2 states this property formally, and because the proof is rather technical we give it in Appendix C.

Theorem 2. *Let $\mathbb{G}_1 = \langle g \rangle, \mathbb{G}_2 = \langle h \rangle$ be groups of order q and let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a non-degenerate bilinear pairing. Let $H : M \times R \to \mathbb{G}_2$ be a hash function with message space M and randomness space R, modelled as a random oracle. Let A be an EUF-CMA adversary against $\mathcal{M}$ who makes q_h queries to the hash function (random oracle) and q_v verification queries, then there exists an adversary B (of similar computational complexity) against the TBDH Problem of the groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ such that:*

$$\mathbf{Adv}^{eufcma}_{\mathcal{M}}(A) \leq q_h(q_v + 1)\mathbf{Adv}^{tbdh}(B) \ .$$

3 Leakage Resilient Key Update

The core idea behind our construction is to secret share the key, and then update the shares independently. In this section we introduce a couple of security definitions for updating key shares in the presence of leakage. We define a key update algorithm as the tuple $\mathcal{KU} = (KG, SHARE, UPDATE1, UPDATE2)$ where KG generates a key, $SHARE$ splits the key into two parts and the update functions are used to update the respective key shares.

We distinguish between two security games: decisional and computational key recovery under leakage (DKR-UL and CKR-UL). In the decisional game (Def. 2), the adversary is given either the real or a fake key and must, based on the leakage emanating from the updates, decide which it is (real or fake). In the computational version (Def. 3), the adversary just gets to see the leakage and must compute the (real) key. After providing the formal definitions, including the corresponding games, we present a concrete construction that meets both notions (in the generic group model).

Definition 2 (Decisional Key Recovery Under Leakage (DKR-UL)). *Let a key update scheme be given. Then the advantage of an adversary A attempting decisional key recovery under key leakage is given by the game in Fig. 2 and $\mathbf{Adv}^{dkrul}(A) = 2\Pr[b' = b : b' \leftarrow A] - 1$.*

Fig. 2 should be understood as follows.[3] The game (representing the environment) itself 'calls' initialise and generates two random keys K_0, K_1 and shares K_0, before returning one of the keys randomly

[3] All other games (and corresponding descriptions in figures) can be understood in a similar way.

to the adversary. The adversary can then call the update function many times, each time giving two leakage functions. The update function then updates the key as defined by the key update scheme before calculating the leakage on the shares and returning this leakage to the adversary. Eventually, the adversary terminates with a guess whether it thinks it holds the key that is being updated (and leaked on) or some unrelated key. The game then 'calls' finalise which checks and declares if the adversary is correct and therefore wins the game.

Def. 3 gives the computational version of the key recovery problem. Like other problems where a decisional and a computational variant exist there is a simple connection between them: an adversary for the computational variant can be used to solve the decisional variant. Consequently, any scheme which is secure against the decisional problem must also be secure against the computational version of the problem, and hence security with regards to DKR-UL implies security with respect to CKR-UL.

Definition 3 (Computation Key Recovery Under Leakage (CKR-UL)). *Let a key update scheme be given. Then the advantage of an adversary A attempting computational key recovery under key leakage is given by the game in Fig. 3 and* $\mathbf{Adv}^{ckrul}_{\mathcal{KU}}(A) = \Pr[A \Rightarrow True]$.

3.1 A Concrete Key Update Construction

Following the approach taken by Kiltz and Pietrzak [15]), we define a concrete Key Update construction $\mathcal{KU} = (KG, SHARE, UPDATE1, UPDATE2)$ in Fig. 4. It is based on using a group and its operation to share and update the key. In order for the CKRUL and DKRUL problems to be hard, it is necessary that the discrete logarithm problem in the group is hard.

Theorem 3. *In the Generic Group Model (see Appendix A) the key update construction $\mathcal{KU}$ is DKR-UL secure, such that:*

$$\mathbf{Adv}^{dkrul}_{\mathcal{KU}}(A) \leq \frac{2^{2\lambda+1} q_u^3}{q}$$

where q_u is the number of update queries the adversary A makes, q is the order of the group and λ is the number of bits that the calls to $UPDATE1, UPDATE2$ are allowed to leak. More concretely for a security parameter of n, we can allow $\lambda = \frac{1}{2}(\log q - 3\log q_u - n)$ bits of leakage.

Proof (sketch). The full proof is rather technical and can be provided in an extended version of this paper. It essentially follows the proof provided by Kiltz and Pietrzak for [15, Lemma 2]. Using their notation, the major difference between the two proofs is that instead of having $\xi(s), \xi(x), \xi(r_0)$ that can have the discrete log target point randomly assigned to them, we only have $\xi(x), \xi(r_0)$. As a direct consequence of this we only get a factor 2 against the discrete log advantage instead of a factor 3. The rest of the proof is similar resulting in the same final bound.

proc *Initialise*():
$K_0 \leftarrow KG$
$K_1 \leftarrow KG$
$b \xleftarrow{\$} \{0,1\}$
$(s_1, s'_1) \leftarrow SHARE(K_0)$
Return K_b

proc *Finalise*(b'):
Return $(b = b')$

proc *update*(f, g):
if Range of f or g is not $\{0,1\}^\lambda$ **then**
  Return $\perp$
end if
$(s_{i+1}, O) \xleftarrow{x} UPDATE1(s_i)$
$\Lambda \leftarrow f(s_i, x)$
$s'_{i+1} \xleftarrow{x'} UPDATE2(s'_i, O)$
$\Lambda' \leftarrow g(s'_i, O, x')$
Return (Λ, Λ')

Fig. 2. The DKR-UL security game.

proc $Initialise()$:
$K \leftarrow KG$
$(s_1, s'_1) \leftarrow SHARE(K)$
Return $e(K, h)$

proc $Finalise(K')$:
Return $(K = K')$

proc $update(f, g)$:
if Range of f or g is not $\{0,1\}^\lambda$ **then**
  Return $\perp$
end if
$(s_{i+1}, O) \xleftarrow{x} UPDATE1(s_i)$
$\Lambda \leftarrow f(s_i, x)$
$s'_{i+1} \xleftarrow{x'} UPDATE2(s'_i, O)$
$\Lambda' \leftarrow g(s'_i, O, x')$
Return (Λ, Λ')

Fig. 3. The CKR-UL security game

proc $KG()$:
$K \xleftarrow{\$} \mathbb{G}_1$
Return K

proc $SHARE(K)$:
$s_1 \xleftarrow{\$} \mathbb{G}_1$
$s'_1 \xleftarrow{\$} K \cdot s_1^{-1}$
Return (s_1, s'_1)

proc $UPDATE1(s_i)$:
$r \xleftarrow{\$} \mathbb{G}_1$
$s_{i+1} \leftarrow s_i \cdot r$
Return (s_{i+1}, r)

proc $UPDATE2(s'_i, r)$:
$s'_{i+1} \leftarrow s'_i \cdot r^{-1}$
Return s'_{i+1}

Fig. 4. The key update construction $\mathcal{KU}$

As the leakage bound λ is smaller than $\frac{\log q}{2}$. the leakage cannot reveal to the adversary the representation of any group element that it could not already compute efficiently without the leakage. This is formalized by considering two leakage functions f_i, f_j with $i < j + 1$ that compute the same element, and using this collision to solve the discrete log problem (which is known to be hard in the Generic Group Model [21]). Consequently, multiple leakage functions will not leak on identical group elements, unless they are already known. Once leakage for individual group elements is restricted to one invocation only, the Generic Group Model ensures sufficient min-entropy is left in its representation (after leakage) to make the only remaining strategy, namely guessing, hard. □

4 A Leakage Resilient MAC

Based on the same principles as the bilinear ElGamal scheme by Kiltz and Pietrzak [15], the MAC scheme and the key update mechanism of the previous two sections turn out to be compatible. The MAC key is a secret group element (as used by the key update mechanism) and, due to the properties of a bilinear pairing, it is possible to compute the tagging algorithm based on the shared key without the need to reconstruct the underlying secret group element. In the next sections we give the construction in more detail, argue why it is resilient against key recovery in practical side channel scenario, and give formal proofs of leakage resilience of the MAC scheme (against both key recovery and existential forgeries) based on the leakage resilience of the key update function (and security of the MAC without leakage).

4.1 The Construction

To make the aforementioned MAC scheme leakage resilient, the key K will be secret shared into shares s and s' such that $s \cdot s' = K$. Since for any W

$$e(s_i, W) \cdot e(s'_i, W) = e(K, W)$$

proc $KG^*()$:
$K \xleftarrow{\$} \mathbb{G}_1$
$s_1 \xleftarrow{\$} \mathbb{G}_1$
$s'_1 \leftarrow K \cdot s_1^{-1}$
Return (s_1, s'_1)

proc $TAG1^*(s_i, m)$:
$w \xleftarrow{\$} R$
$W \leftarrow H(m, w)$
$t_i \leftarrow e(s_i, W)$
$r_i \xleftarrow{\$} \mathbb{G}_1$
$s_{i+1} \leftarrow s_i \cdot r_i$
Return (t_i, W, w, r_i)

proc $TAG2^*(s'_i, t_i, W, w, r_i)$:
$t'_i \leftarrow e(s'_i, W)$
$s'_{i+1} \leftarrow s'_i \cdot r_i^{-1}$
$T \leftarrow t_i \cdot t'_i$
Return (T, w)

proc $VRFY1^*(s_i, (T, w), m)$:
$W \leftarrow H(m, w)$
$t_i \leftarrow e(s_i, W)$
$r_i \xleftarrow{\$} \mathbb{G}_1$
$s_{i+1} \leftarrow s_i \cdot r_i$
Return (t_i, W, w, r)

proc $VRFY2^*(s'_i, T, t_i, W, w, r_i)$:
$t'_i \leftarrow e(s'_i, W)$
$s'_{i+1} \leftarrow s'_i \cdot r_i^{-1}$
$T' \leftarrow t_i \cdot t'_i$
Return $(T' = T)$

Fig. 5. Leakage Resilient MAC $\mathcal{M}^*$

this allows the tagging and verification algorithms to be split into two, each of which taking a share of the key. As the key update function maintains as invariant that $s_i \cdot s'_i = K$ for all rounds i (possibly independently for tagger and multiple verifiers), correctness of the scheme follows as before. We define our MAC $\mathcal{M}^* = (KG^*, TAG1^*, TAG2^*, VRFY1^*, VRFY2^*)$ in Fig. 5.

To prove any form of leakage resilience we will use the "Only Computation Leaks Information" assumption. For this purpose we have explicitly split the tagging and verification algorithms in two (according to which share is being operated upon). This OCLI assumption then implies that the two parts of the tagging (resp. verification) algorithms will only leak on their respective shares (independent of the other share). As a result, direct leakage on the underlying key K is not possible. This turns out sufficient for formal proofs of security.

Some considerations with regards to side channel attacks In practice, the primary concern is retrieval of the secret key due to side channel leakage of the secret key. Since the key is shared and updated each round, the main worry is leakage of K within a single call to the tagging (or verification) algorithm. Even though in reality the two parts of the tagging algorithm will be implemented on the same device, the assumption that the various computations i) do not interfere with each other, and ii) are the main source of leakage, seems experimentally justified at this high level of computation. We only consider side channel attacks using a single side channel observation, as the key updates thwarts any differential style attacks.

Noting that leakage of the generation of the randomness w and the computation of W using the hash function can be ignored (all values involved are public), there are 6 computations part of the tagging algorithm that could leak. We note that leakage of the partial tags is unlikely to aid in key recovery.

- Generation of the key update mask r_i. If a mask would completely leak, this would theoretically allow linking leakage from one invocation of the tagging algorithm to the next: if an adversary learns r_i he could use this knowledge to let the leakage on s_{i+1} sneakily be leakage on s_i instead, obtaining more leakage on the share s_i than should be allowed. In practice however, it is unclear whether knowing r_i really allows leakage on the share s_{i+1} to be meaningfully correlated to the share s_i.
- (Twice) Updating the secret key shares. Since this operation uses the key shares directly, full leakage of the previous share or the next share is clearly fatal.
- (Twice) Computing the tag. The underlying computation is a pairing evaluation, where one of the operands is known, the other is secret. Complete leakage on the secret data again leads to breaking the system.

adversary $B(V = e(K, h))$:
$K' \leftarrow A^{TagSIM, VRFYSIM}$
Return K'

sim $HSIM(m, w)$:
if $W[m][w] = \perp$ **then**
  $W[m][w] \leftarrow \mathbb{Z}_q$
end if
Return $h^{W[m][w]}$

sim $TagSim(m, f, g)$
$w \xleftarrow{\$} R$
if $W[m][w] = \perp$ **then**
  $W[m][w] \leftarrow \mathbb{Z}_q$
end if
$W \leftarrow h^{W[m][w]}$
Create f', g' by fixing W, w, m in f, g
$(F, G) \leftarrow Update(f', g')$
$T \leftarrow V^{W[m][w]}$
Return $((T, w), F, G)$

sim $VRFYSim((T, w)m, f, g)$
if $W[m][w] = \perp$ **then**
  $W[m][w] \leftarrow \mathbb{Z}_q$
end if
$W \leftarrow h^{W[m][w]}$
Create f', g' by fixing W, w, m in f, g
$(F, G) \leftarrow Update(f', g')$
$T' \leftarrow V^{W[m][w]}$
$b \leftarrow (T' = T)$
Return (b, F, G)

Fig. 6. An adversary against the CKR-CMLA security of $\mathcal{M}^*$ being used to build an adversary against the CKR-UL security of $\mathcal{KU}$

– Recombining the parts of the tag into a single tag. This is done by a single multiplication in the group $\mathbb{G}_T$. Because this computation is no longer depending directly on the key (or its shares), we do not consider this operation a target for an attacker.

Clearly, we require some form of protection against simple side channel attacks (e.g. SPA) in any implementation of our scheme. However, as we show our scheme is leakage resilient (up to some bound), we can allow a modest amount of leakage, implying that the SPA countermeasures need not be watertight.

Key recovery under chosen message attacks with leakage (CKR-CMLA). In the paragraph above, we already touched on key recovery from a practical perspective and observed that an adversary will most likely be left empty handed provided some techniques to prevent full key exposure (via e.g. SPA) are implemented. Here we prove that this indeed the case using the formal framework of leakage resilience.

The advantage of proving that $\mathcal{M}^*$ is CKR-CMLA in a modular manner (instead of proving it with a direct reduction) is that although the current justification (of the key update's leakage resilience) relies on the generic group model, explicitly surfacing the CKR-UL assumption allows a more focussed study on the security of our scheme, and comparison of groups that might be suitable to instantiate it.

Theorem 4. *Let A be an adversary against the CKR-CMLA security of $\mathcal{M}^*$ making q queries in total to its tagging and verification oracles. Then there exists an (equally efficient) adversary B against the CKR-UL security of $\mathcal{KU}$ that makes q key update queries and for which*

$$\mathbf{Adv}^{ckrcmla}_{\mathcal{M}^*}(A) \leq \mathbf{Adv}^{ckrul}_{\mathcal{KU}}(B)$$

Proof. Assume we have an adversary A against the CKR-CMLA security of $\mathcal{M}^*$, then Fig. 6 shows how to construct an adversary B against the CKR-UL security of $\mathcal{KU}$.

It can be verified that whenever A extracts the key, B also extracts the key and thus $\mathbf{Adv}^{ckrul}_{\mathcal{M}^*}(A) \leq \mathbf{Adv}^{ckrul}_{\mathcal{KU}}(B)$ as required. □

4.2 Existential Unforgeability against Chosen Message Attacks with Leakage

Definition 4 gives a modification of the usual game for existential unforgability under chosen message attacks to a scenario incorporating leakage. (Fig. 9 in Appendix D gives the game instantiated with our specific scheme.) The reason that, in verify, we have the check that this is the first time that a message,

proc $Initialise()$:
$(s, s') \xleftarrow{\$} KG^*$
$V \leftarrow \{\}$
$S \leftarrow \{\}$
$win \leftarrow False$

proc $Finalise((T, w), m)$:
$O \leftarrow VRFY1^*(s, (T, w), m)$
$b \leftarrow VRFY2^*(s', O)$
if $m \notin S$ *and* $b = 1$ **then**
  $Win \leftarrow True$
end if
Return Win

proc $Tag(m, l_1, l_2)$
$S \leftarrow S \cup \{m\}$
$\Lambda_1 \leftarrow l_1(s, x)$
$O \leftarrow TAG1^*(s, m)$
$\Lambda_2 \leftarrow l_2(s', O, x')$
$(T, w) \leftarrow TAG2^*(s', O)$
Return $((T, w), \Lambda_1, \Lambda_2)$

proc $Verify((T, w), m, l_1, l_2)$
if $(m, w) \in V$ **then**
  Return $\perp$
end if
$V \leftarrow V \cup \{(m, w)\}$
$\Lambda_1 \leftarrow l_1(s, x)$
$O \leftarrow VRFY1^*(s, (T, w), m)$
$\Lambda_2 \leftarrow l_2(s', O, x')$
$b \leftarrow VRFY2^*(s', O)$
Return $(b, \Lambda_1, \Lambda_2)$

Fig. 7. The EUF-CMLA' security game

randomness pair is being checked is otherwise a trivial attack arises in which the attacker checks the same incorrect tag, against the same message, repeatedly each time leaking a bit of correct tag until they have the whole tag and thus a valid forgery. It would be possible to define a stronger notion (EUF-CMLA) where this restriction was not in place and then would have to strengthen the verification step, with some form of leakage resilient equality check but we currently leave this as future research.

l_1, l_2 are the two leakage functions and each one is allowed to leak λ bits. We have $l_1(s, x), l_2(s', O, x')$ where s, s' are the two shares of the key, x, x' is the internal randomness used by the two algorithms and O is the output of $TAG1^*$. We have chosen to have the adversary define l_1, l_2 at the same time and not have the ability to define l_2 after seeing the output of l_1 as this conforms with what other people have been doing.

Definition 4 (EUF-CMLA'). *For adversary A, we define their advantage, of winning the game in Figure 7, as* $\mathbf{Adv}^{eufcmla'}_{\mathcal{M}^*}(A) = \Pr[win = True]$.

Our scheme can be shown to be EUF-CMLA secure provided that the key update scheme is secure against the decisional version of the key recovery game (and of course, the MAC needs to be EUF-CMA without taking any leakage into account. This is captured by Theorem 5, whose proof can be found in Appendix D.

Theorem 5. *Let A be an probabilistic polynomial time adversary against the EUF-CMLA' security of* $\mathcal{M}^*$*, then there exists two probabilistic polynomial time adversaries B and C against the EUF-CMA security of* $\mathcal{M}$ *and the DKR-UL security of* $\mathcal{M}^*$ *respectively such that:*

$$\mathbf{Adv}^{eufcmla'}_{\mathcal{M}^*}(A) \leq \mathbf{Adv}^{eufcma}_{\mathcal{M}}(B) + \mathbf{Adv}^{dkrul}_{\mathcal{M}^*}(C)$$

Where both B and C have a similar query and time cost as A.

5 Practical Aspects of our Scheme

Now that we have our scheme defined and argued about its' security we want to focus our attention towards some practical considerations: how efficient is it in comparison to other leakage resilient MAC constructions, and what would a practical implementation need to guarantee to meet our leakage bound/assumptions?

To provide any actual numbers we need to make some choices for parameters of the various schemes we are comparing. In case of schemes which have as underlying primitive a pseudo random function

(PRF), we chose to instantiate this PRF with AES-128. This is motivated by the fact that this reflects the current state of the art. For our own scheme, and a somewhat comparable signature scheme, we use as instantiation of the bilinear map a pairing defined over a suitable pairing friendly elliptic curve. In this case we choose as group size parameter 2^{160}, again with the motivation to reflect a state-of-the art security bound (i.e. 2^{80} is regarded as minimum for bound which is implied by a group of double this size).

We first provide a comparison with regards to the efficiency of our scheme.

5.1 Comparison with other schemes

In our comparison we essentially look at the number of group operations that tagging and verification require. We also report on the tolerated leakage, the key size and the tag size.

The only other leakage resilient MAC scheme in the literature is [12] to the best of our knowledge. The advantage of [12] is that it only relies on very minimal assumptions (the existence of one-way functions). However the leakage bound is a total bound, i.e. it holds regardless of the number of times the tag and verify algorithms are called. Assuming that we instantiate the PRF required for this scheme with AES-128 (i.e. setting $\lambda = s = 128$ to use the notation in the paper), and using the equations they give in Theorem 5.6, it turns out that AES will be called 512 times per tag and verify query. While this already makes the scheme computationally expensive, the larger problem is the overall key size: the key must be of size approximately 2^{18} bits, which is impractical for embedded devices for which our MAC is aimed at. Even worse, under these parameters the MAC can leak 512 bits over the lifetime of the system (i.e. not per MAC invocation).

As [12] is the only other leakage resilient MAC we are aware of, we also provide a comparison against PRFs and signatures because although there is not an immediate strategy to convert them into Leakage Resilient MACs available, there is potential that one might use PRFs or signatures to instantiate a MAC with some leakage resilient properties.

The PRF by Dodis and Pietrzak [6] requires that the leakage functions are fixed prior to the attack and are not adaptively chosen. They define the PRF $\Gamma^F : \Sigma^{3k+n} \times \Sigma^m \rightarrow \Sigma^{4k+2n}$ created from a wPRF $F : \Sigma^k \times \Sigma^n \rightarrow \Sigma^{4k+2n}$. If we instantiate F with AES-128 with something like $F(x) = Enc_K(x||000)||Enc_K(x||001)||Enc_K(x||010)||Enc_K(x||011)||Enc_K(x||100)||Enc_K(x||101)$ and then take the desired number of output bits, this gives us $k = m = 128$, $n = 125$ and $\Gamma^F : \Sigma^{509} \times \Sigma^{128} \rightarrow \Sigma^{768}$. As stated in the paper each time the PRF is called it calls F m+1 times and thus AES will be called 774 times. Hence even if this can be converted into a MAC reasonably inexpensively, it will still have a high usage cost on a smart card.

The PRF by Faust et al. [8] requires that neither the leakage or the PRF inputs are queried adaptively but are fixed prior to the start of the game. They construct a scheme $\Gamma^{F,m} : \{0,1\}^{k+(m+1)l} \times \{0,1\}^m \rightarrow \{0,1\}^n$ which uses the wPRF $F : \{0,1\}^k \times \{0,1\}^l \rightarrow \{0,1\}^{2k}$ and $m+1$ public random values of length l. If we again instantiated the wPRF with AES-128 to get $F(x) = Enc_K(x||0)||Enc_K(x||1)$ meaning we get $m = k = 128$, $l = 127$,$n = 256$ and $\Gamma^{F,128} : \{0,1\}^{16511} \times \{0,1\}^{128} \rightarrow \{0,1\}^{256}$. This will call AES 258 times per invocation, which while an improvement over the previous scheme, is under stronger assumptions and 258 AES invocations may still be impractical on a smartcard.

The signature scheme by Faust et al. [7] uses $2l$ exponentiations, $4(l-1)$ multiplications, $l-1$ additions and 2 hash function calls in the signing algorithm and tl exponentiations, tl multiplications and t hash function calls in the verification algorithm, where l is related to the underlying l-representation problem [3] which we assume is hard and t is the depth of the signature chain. The downside of this scheme is that even if $l = 2$, while signing is efficient, verification takes longer depending how deep the signature chain is. This could mean that verification quickly becomes too expensive for an embedded device to perform.

The signature scheme by Galindo and Vivek [10] is the only one comes close to our construction in terms of performance, which is unsurprising given that it is also based on [15]. For signing, the algorithm uses 2 Elliptic Curve scalar multiplications, 5 Elliptic Curve additions and also generates a random curve point (and its' inverse), while verification uses 2 Elliptic Curve point additions, 1 Elliptic Curve scalar multiplication and 2 pairings. Since a pairing is currently only slightly more expensive than an exponentiation, our tagging algorithm will be almost equivalent in timing to their signing algorithm. Their verification however is faster than ours. Note that their keys however are larger. Another thing of note, is that while there is no hash function explicit in the scheme, it is assumed that the message comes from $\mathbb{Z}_p$ and thus in practice to sign arbitrary messages, the message would have to be hashed onto $\mathbb{Z}_p$.

Table 1. A comparison of possible EUF-CMLA schemes

Scheme	Leakage	Key Size	Tag Size	Tag Time	Verification Time
Our scheme	$\frac{1}{2}(\log q - 3\log q_u - n)$ see Theorem 3 (approx 60 bits)	2 EC points (approx 320 bits)	1 EC point 1 random string (approx 228 bits)	1 hash 1 random point (and inverse) generated 3 EC additions 2 pairings	1 hash 1 random point (and inverse) generated 3 EC additions 2 pairings
HLWW:[12]	512 bits total	approx 2^{18} bits	approx 2^{19} bits	512 AES calls	512 AES calls
DP:[6]	$\frac{\log(\epsilon^{-1})}{6}$ for $\epsilon = \mathbf{Adv}_F^{wprf}(A)$ (approx 22 bits)	509 bits	762 bits	774 AES calls	774 AES calls
FPS:[8]	$\frac{\log(\epsilon^{-1})}{4}$ for $\epsilon = \mathbf{Adv}_F^{wprf}(A)$ (approx 32 bits)	128 bits 16,383 public bits	256 bits	258 AES calls	258 AES calls
FKPR:[7]	$2\text{kl}(\frac{1}{2} - \frac{1}{2l} - \delta)$ (approx 40 bits)	Varies	Varies	$2l$ exponentiations $4(l-1)$ multiplications $l-1$ additions 2 hashes	Varies
GV:[10]	$<< \frac{\log q}{2}$ q is group size (approx 60 bits)	sk: 2 EC points pk: 3 EC points (approx 800 bits)	2 EC points (approx 320 bits)	1 random point (and inverse) generated 2 scalar multiplications 5 EC additions 1 hash	1 scalar multiplication 2 EC additions 2 pairings 1 hash

5.2 Security considerations

The leakage bound we have for our scheme is that we can tolerate up to approx. 60 bits of leakage assuming a group size of about 2^{160} per invocation of the scheme. For a practical implementation it is important to acknowledge that this will not include the initial sharing out of the key. Consequently in a strict sense this would need to be done in a secure environment.

When considering what '60' bits of leakage means given our constructions, it helps to think about the tagging and verification algorithms in a concrete instantiation: i.e. we are working with pairings that are defined over some pairing friendly curve. Consequently, s_i, s'_i, r_i and ${r_i}^{-1}$ are elliptic curve points, $s_i \cdot r_i$ and $s_i \cdot {r_i}^{-1}$ are elliptic curve point additions, and $e()$ is a pairing. We have already run through some security considerations in the previous section. Now that we have a concrete instantiation of our scheme,

we can argue more concretely about implementation options. In order to provide some resistance against simple side channel attacks such as SPA we would hence essentially need to blind the EC points which correspond to secrets. A recent contribution [2] gives a good overview on various side channel attacks on pairings from which one can again conclude that also the pairing operation should best be implemented on blinded EC points.

The obvious question arising now might be what we have gained in practice, as we yet again need to apply the basic techniques that would prevent attacks such as SPA and DPA? The answer is the non leakage resilient version, e.g. even partial leakage on the random values will probably allow lattice based attacks, e.g [20]. The leakage resilient scheme however guarantees that in the presence of partial leakage no such attack can succeed.

6 Acknowledgements

Dan Martin and Elisabeth Oswald have been supported in part by EPSRC via grant EP/I005226/1.

References

1. J. Balasch, S. Faust, B. Gierlichs, and I. Verbauwhede *Theory and Practice of a Leakage Resilient Masking Scheme.* Asiacrypt 2012, pp. 758–775, LNCS 7658, Springer, 2012
2. Johannes Blömer, Peter Günther, Gennadij Liske *Improved Side Channel Attacks on Pairing Based Cryptography.* IACR Cryptology ePrint Archive, 2011
3. S. Brands, *An Efficient Off-Line Electronic Cash System Based on the Representation Problem.* Technical Report CS-R9323, CWI, Amsterdam, 1993.
4. D. Coppersmith, *Finding a Small Root of a Univariate Modular Equation.* Eurocrypt 1996.
5. S. Dziembowski, K. Pietrzak, *Leakage-Resilient Cryptography.* 49th FOCS, pp293-302. IEE Computer Society Press, 2008.
6. Y. Dodis, K. Pietrzak, *Leakage-Resilient Pseudorandom Functions and Side-Channel Attacks on Feistel Networks.* CRYPTO 2010.
7. S. Faust, E. Kiltz, K. Pietrzak, G. Rothblum, *Leakage-Resilient Signatures.* Cryptology ePrint Archive, Report 2009/282. 2009.
8. S. Faust, K Pietrzak, J. Schipper, *Practical Leakage-Resilient Symmetric Cryptography.* CHES 2012, pp 213-232, 2012.
9. S. Galbraith, K. Paterson, N. Smart. *Pairings for Cryptographers.* Discrete Appl. Math., 156(2008), pp 3113-3121. 2008.
10. D. Galindo, S. Vivek, *A Practical Leakage-Resilient Signature Scheme in the Generic Group Model.* SAC 2012. 2012.
11. J. Dj. Golic, C. Tymen, *Multiplicative Masking and Power Analysis of AES.* CHES 2002, pp. 198–212 2002.
12. C. Hazay, A.López-Alt, H. Wee, D. Wichs, *Leakage-Resilient Cryptography from Minimal Assumptions.* Cryptology ePrint Archive, Report 2012/604. 2012.
13. P. Kocher, *Leak Resistant Cryptographic Indexed Key Update.* US Patent 6539092.
14. P.C. Kocher, J. Jaffe, B. Jun, *Differential power analysis.* CRYPTO 1999, LNCS 1666, pp. 388—397, Springer 1999.
15. E. Kiltz, K. Pietrzak, *Leakage Resilient ElGamal Encryption.* AsiaCrypt 2010, LNCS, vol 6477, pp 595-612, 2010.
16. M. Medwed, F.-X. Standaert, J. Groszschadl, F. Regazzoni, *Fresh Re-keying: Security against Side-Channel and Fault Attacks for Low-Cost Devices.* AFRICACRYPT 2010, pp. 279–296, LNCS 6055, Springer, 2010.
17. S. Micali, L. Reyzin, *Physically Observable Cryptography.* TCC 2004, LNCS, vol 2951, pp 278-296, Cambridge, MA, USA, February 2004.
18. U. Maurer *Abstract Models of Computation in Cryptography.* 10th IMA Conference on Cryptography and Coding, LNCS 2796, pp 1-12. 2005.
19. V. Nechaev, *Complexity of a Determinate Algorithm for the Discrete Logarithm.* Mathematical Notes 55(2) pp165-174, 1994.
20. P. Nguyen, I. Shparlinsk, *The Insecurity of the Elliptic Curve Digital Signature Algorithm with Partially Known Nonces.* Des. Codes Cryptography 30(2): 201-217 (2003)
21. V. Shoup, *Lower Bounds for Discrete Logarithms and Related Problems.* Eurocrypt 1997, LNCS, vol 1233, pp 256-266. 1997.
22. Y Yacobi, *A Note on the Bilinear Diffie Hellman Assumption.* Cryptology ePrint Archive, Report 2004/308. 2004.

A Generic Bilinear Groups

In the Generic Group model [19],[21],[18] each element of the group is represented by a unique random string. As a side effect of this the only ability the adversary has given two group elements is to test for equality.

In the Generic Bilinear Group (GBG) model each of the two or three (dependant on if using a symmetric pairing or not) has its own randomised encoding. Each of these encodings will be represented by an injective encoding function $\xi_1 : \mathbb{Z}_q \to \Xi_1, \xi_2 : \mathbb{Z}_q \to \Xi_2, \xi_T : \mathbb{Z}_q \to \Xi_T$ for $\mathbb{G}_1, \mathbb{G}_2, \mathcal{G}_T$ respectively. Since the adversary needs to perform more operations that just equality checks on elements he has access to the following 4 oracles:

- $\mathcal{O}_1(\xi_1(a), \xi_1(b)) = \xi_1(a + b \mod q)$
- $\mathcal{O}_2(\xi_2(a), \xi_2(b)) = \xi_2(a + b \mod q)$
- $\mathcal{O}_T(\xi_T(a), \xi_T(b)) = \xi_T(a + b \mod q)$
- $\mathcal{O}_e(\xi_1(a), \xi_2(b)) = \xi_T(a \cdot b \mod q)$

for all $a, b \in \mathbb{Z}_q$. Each of the 4 oracles will return $\perp$ if either of the inputs is an invalid encoding of an underlying group element. $\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_T$ perform the group operations of $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ respectively, while $\mathcal{O}_e$ performs the pairing operation. To work with these groups an adversary only needs to be given $\xi_1(1), \xi_2(1)$ and access to the four oracles, from which any group elements can be computed.

B More Diffie-Hellman Problems

Definition 5 (Target Bilinear Diffie-Hellman with Oracle Problem). *Let $\mathbb{G}_1 = \langle g \rangle, \mathbb{G}_2 = \langle h \rangle, \mathbb{G}_T$ be three groups of order q with the non-degenerate bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ between them. We say the Target Bilinear Diffie-Hellman with Oracle (TBDHwO) Problem is hard if given h^x, f^y it is hard to compute f^{xy}, when you have access to an oracle test(X) that tells you if $X = f^{xy}$, where x, y are sampled uniformly at random from $\mathbb{Z}_q$. Given an adversary A we define their advantage of winning this game as* $\mathbf{Adv}^{tbdhwo}(A) = \Pr[\mathcal{A} = f^{xy} : \mathcal{A} \leftarrow A^{test}(g, h, h^x, f^y)]$

Lemma 1. *Assuming the TBDH problem is hard then the TBDHwO problem is also hard. Moreover* $\mathbf{Adv}^{tbdhwo}(A) \leq (q_t + 1)\mathbf{Adv}^{tbdh}(B)$*, where q_t is the number of test queries made.*

This is straightforward to see since every time A wants a *test* query answered, instead of asking the *test* oracle, he runs a copy of his algorithm and outputs the value he wants testing and based on if this copy wins or not tells him what value the test function call will return. Since this needs to be done for each test function the bound given holds.

C EUF-CMA Security Proof for $\mathcal{M}$

To prove this statement we will show if there is an adversary A who can construct a valid forgery on a message, using q_h queries to the Random Oracle, then this can be used to construct an adversary B who can solve the TBDHwO in these groups. We can assume without loss of generality that if A creates the forgery (T, w) on m that it queried $H(m, w)$ and that each (m, w) pair is only queried once. The reduction is then given in Figure 8.

adversary $B(\mathcal{H} = h^x, \mathcal{F} = f^y)$:
$i \leftarrow 0$
$j \xleftarrow{\$} [q_h]$
$((T, w), m) \leftarrow A()$
Return T

sim $TagSIM(m)$:
$w \xleftarrow{\$} R$
if $W[m][w] = \perp$ **then**
  $W[m][w] \xleftarrow{\$} \mathbb{Z}_q$
else if $W[m][w] = flag$ **then**
  $ABORT$
end if
$T \leftarrow \mathcal{F}^{W[m][w]}$
Return T

sim $VRFYSIM((T, w), m)$:
if $W[m][w] = \perp$ **then**
  $W[m][w] \xleftarrow{\$} \mathbb{Z}_q$
else if $W[m][w] = flag$ **then**
  Return $test(T)$
end if
Return $(T = \mathcal{F}^{W[m][w]})$

sim $HSIM(m, w)$:
$i \leftarrow i + 1$
if $W[m][w] = \perp$ **then**
  if $i = j$ **then**
    $W[m][w] \leftarrow flag$
    Return $\mathcal{H}$
  else
    $W[m][w] \xleftarrow{\$} \mathbb{Z}_q$
  end if
end if
Return $h^{W[m][w]}$

Fig. 8. Constructing a TBDHwO Adversary from a EUF-CMA Adversary

proc $Initialise()$:
$K \xleftarrow{\$} \mathbb{G}_1$
$s \xleftarrow{\$} \mathbb{G}_1$
$s' \leftarrow K \cdot s^{-1}$
$V \leftarrow \{\}$
$S \leftarrow \{\}$
$win \leftarrow False$

proc $Finalise((T, w), m)$:
$(T, w, W, t, r) \leftarrow VRFY1^*(s, (T, w), m)$
$b \leftarrow VRFY2^*(s', T, t, W, w, r, x')$
if $m \notin S$ *and* $b = 1$ **then**
  $Win \leftarrow True$
end if
Return Win

proc $Tag(m, l_1, l_2)$
$S \leftarrow S \cup \{m\}$
$\Lambda_1 \leftarrow l_1(s)$
$(W, w, t, r) \leftarrow TAG1^*(s, m)$
$\Lambda_2 \leftarrow l_2(s', W, w, t, r)$
$(T, w) \leftarrow TAG2^*(s', W, w, t, r)$
Return $((T, w), \Lambda_1, \Lambda_2)$

proc $Verify((T, w), m, l_1, l_2)$
if $(m, w) \in V$ **then**
  Return $\perp$
end if
$V \leftarrow V \cup \{(m, w)\}$
$\Lambda_1 \leftarrow l_1(s, x)$
$(T, w, W, t, r) \leftarrow VRFY1^*(s, (T, w), m)$
$\Lambda_2 \leftarrow l_2(s', W, w, t, r)$
$b \leftarrow VRFY2^*(s', T, t, W, w, r, x')$
Return $(b, \Lambda_1, \Lambda_2)$

Fig. 9. The EUF-CMLA security game instantiated with $\mathcal{M}^*$

From this we can see that when the message, randomness pair forged on was the j^{th} query to the RO, B has won the TBDH game. This is because this game has an implied key of $K = g^y$ and thus $T = e(K, H(m, w))$ by the definition of $\mathcal{M}$ and $H(m, w) = \mathcal{H}$ when they forge on the j^{th} query, giving $e(K, \mathcal{H}) = e(g^y, h^x) = f^{xy}$, breaking the TBDH game.

This happens with probability $\frac{1}{q_h}$. The $ABORT$ does not affect this probability, since the only time B will abort is if A tries to tag on the value B wants it to make a forgery on and hence a forgery on this value is no longer possible. Thus $\mathbf{Adv}^{eufcma}_{\mathcal{M}}(A) \leq q_h \cdot \mathbf{Adv}^{tbdhwo}(B)$ and by Lemma 1 the theorem holds. Since $\mathbf{Adv}^{tbdhwo}(B)$ is negligible by the assumption that the TBDH problem is hard in these groups we get that $\mathcal{M}$ is a UF-CMA secure MAC as required. □

D EUF-CMLA' Security Proof for $\mathcal{M}^*$

Claim ($\Pr[G_0 \Rightarrow 1] = \mathbf{Adv}^{eufcmla'}_{\mathcal{M}^*}(A)$*).* The advantage of an adversary winning G_0 (see Figure 10) is the same as them winning the EUF-CMLA game, since the only difference is that initialise generates a second key K' randomly which is never used or seen by the adversary.

Claim ($\Pr[G_1 \Rightarrow 1] = \Pr[G_0 \Rightarrow]$*).* In game G_1 (see Figure 10) $TAG2^*$ is replaced by $e(K, W)$ and $VRFY2^*$ is replaced with $(T = e(K, W))$, that is they are replaced with the non-shared version. As shown when $\mathcal{M}^*$ was introduced these are equivalent and since we do not change the leakage in this game, we have the advantage of winning this game is the same advantage as winning G_0.

Claim ($\Pr[G_1 \Rightarrow 1] - \Pr[G_2 \Rightarrow 1] \leq \mathbf{Adv}^{dkrul}_{\mathcal{M}^*}(C)$ *).* In game G_2 (see Figure 11) it is now K' that is shared and updated, instead of K. However due to the changes made in the previous game, all the tags are still calculated using K while the leakage is calculated on K'. If the adversary can tell the difference between the two games he can solve the DKR-UL game, which is secure by assumption.

Claim. Let A be a PPT adversary who can win at game G_2, the there exists a PPT adversary B against the EUF-CMA security of $\mathcal{M}$, such that:

$$\Pr[G^A_0 \Rightarrow 1] \leq \mathbf{Adv}^{eufcma}_{\mathcal{M}}(B)$$

Proof (of claim). Let A be an adversary who can win at game G_2, we will show, in Figure 12 how to use it to construct an adversary B who can win the EUF-CMA game against $\mathcal{M}$.

From this we get $\Pr[G^A_0 \Rightarrow 1] \leq \mathbf{Adv}^{eufcma}_{\mathcal{M}}(B)$ such that A and B have the same query complexity. This is because all of the leakage is computed on a random key which can be chosen and computed by B, while all the Tags are generated by B calling its tag and verify oracles, thus whenever A can create a forgery, so can B. △

Putting all of this together we have:

$$\begin{aligned}
\mathbf{Adv}^{eufcmla'}_{\mathcal{M}^*}(A) &= \Pr[G_0 \Rightarrow 1] \\
&= \Pr[G_1 \Rightarrow 1] \\
&\leq \Pr[G_2 \Rightarrow 1] + \mathbf{Adv}^{dkrul}_{\mathcal{M}^*}(C) \\
&\leq \mathbf{Adv}^{eufcma}_{\mathcal{M}}(B) + \mathbf{Adv}^{dkrul}_{\mathcal{M}^*}(C)
\end{aligned}$$

as required. □

Game G_0

proc $Initialise()$:
$K \xleftarrow{\$} \mathbb{G}_1$
$K' \xleftarrow{\$} \mathbb{G}_1$
$s \xleftarrow{\$} \mathbb{G}_1$
$s' \leftarrow K \cdot s^{-1}$
$V \leftarrow \{\}$
$S \leftarrow \{\}$
$win \leftarrow False$

proc $Tag(m, l_1, l_2)$
$S \leftarrow S \cup \{m\}$
$\Lambda_1 \leftarrow l_1(s)$
$(W, w, t, r) \leftarrow TAG1^*(s, m)$
$\Lambda_2 \leftarrow l_2(s', W, w, t, r)$
$(T, w) \leftarrow TAG2^*(s', W, w, t, r)$
Return $((T, w), \Lambda_1, \Lambda_2)$

proc $Verify((T, w), m, l_1, l_2)$
if $(m, w) \in V$ **then**
  Return $\perp$
end if
$V \leftarrow V \cup \{(m, w)\}$
$\Lambda_1 \leftarrow l_1(s, x)$
$(T, w, W, t, r) \leftarrow VRFY1^*(s, (T, w), m)$
$\Lambda_2 \leftarrow l_2(s', W, w, t, r)$
$b \leftarrow VRFY2^*(s', T, t, W, w, r, x')$
Return $(b, \Lambda_1, \Lambda_2)$

proc $Finalise((T, w), m)$:
$(T, w, W, t, r) \leftarrow VRFY1^*(s, (T, w), m)$
$b \leftarrow VRFY2^*(s', T, t, W, w, r, x')$
if $m \notin S$ *and* $b = 1$ **then**
  $Win \leftarrow True$
end if
Return Win

Game G_1

proc $Initialise()$:
$K \xleftarrow{\$} \mathbb{G}_1$
$K' \xleftarrow{\$} \mathbb{G}_1$
$s \xleftarrow{\$} \mathbb{G}_1$
$s' \leftarrow K \cdot s^{-1}$
$V \leftarrow \{\}$
$S \leftarrow \{\}$
$win \leftarrow False$

proc $Tag(m, l_1, l_2)$
$S \leftarrow S \cup \{m\}$
$\Lambda_1 \leftarrow l_1(s)$
$(W, w, t, r) \leftarrow TAG1^*(s, m)$
$\Lambda_2 \leftarrow l_2(s', W, w, t, r)$
$T \leftarrow e(K, W)$
Return $((T, w), \Lambda_1, \Lambda_2)$

proc $Verify((T, w), m, l_1, l_2)$
if $(m, w) \in V$ **then**
  Return $\perp$
end if
$V \leftarrow V \cup \{(m, w)\}$
$\Lambda_1 \leftarrow l_1(s, x)$
$(T, w, W, t, r) \leftarrow VRFY1^*(s, (T, w), m)$
$\Lambda_2 \leftarrow l_2(s', W, w, t, r)$
$b \leftarrow (T = e(K, W))$
Return $(b, \Lambda_1, \Lambda_2)$

proc $Finalise((T, w), m)$:
$(T, w, W, t, r) \leftarrow VRFY1^*(s, (T, w), m)$
$b \leftarrow (T = e(K, W))$
if $m \notin S$ *and* $b = 1$ **then**
  $Win \leftarrow True$
end if
Return Win

Fig. 10. Games G_0 and G_1 used in the EUF-CMLA security proof of $\mathcal{M}^*$

Game G_2

proc $Initialise()$:
$K \xleftarrow{\$} \mathbb{G}_1$
$K' \xleftarrow{\$} \mathbb{G}_1$
$s \xleftarrow{\$} \mathbb{G}_1$
$s' \leftarrow K' \cdot s^{-1}$
$V \leftarrow \{\}$
$S \leftarrow \{\}$
$win \leftarrow False$

proc $Finalise((T, w), m)$:
$b \leftarrow (T = e(K, W))$
if $m \notin S$ *and* $b = 1$ **then**
  $Win \leftarrow True$
end if
Return Win

proc $Tag(m, l_1, l_2)$
$S \leftarrow S \cup \{m\}$
$\Lambda_1 \leftarrow l_1(s)$
$(W, w, t, r) \leftarrow TAG1^*(s, m)$
$\Lambda_2 \leftarrow l_2(s', W, w, t, r)$
$(T, w) \leftarrow e(K, W)$
Return $((T, w), \Lambda_1, \Lambda_2)$

proc $Verify((T, w), m, l_1, l_2)$
if $(m, w) \in V$ **then**
  Return $\perp$
end if
$V \leftarrow V \cup \{(m, w)\}$
$\Lambda_1 \leftarrow l_1(s, x)$
$(T, w, W, t, r) \leftarrow VRFY1^*(s, (T, w), m)$
$\Lambda_2 \leftarrow l_2(s', W, w, t, r)$
$b \leftarrow (T = e(K, W))$
Return $(b, \Lambda_1, \Lambda_2)$

Fig. 11. Game G_2 used in the EUF-CMLA security proof of $\mathcal{M}^*$

adversary $B()$:
$K' \xleftarrow{\$} \mathcal{G}_1$
$s_0 \xleftarrow{\$} \mathcal{G}_1$
$s'_0 \leftarrow K' \cdot s_0^{-1}$
$(T, w) \leftarrow A^{TagSIM, VRFYSIM}$
Return (T, w)

sim $TagSIM(m, l_1, l_2)$:
$(T, w) \leftarrow Tag(m)$
$W \leftarrow H(m, w)$
$r \xleftarrow{\$} \mathcal{G}_1$
$t_i \leftarrow e(s_i, W)$
$\Lambda_i \leftarrow l_1(s_i)$
$\Lambda'_i \leftarrow l_2(s'_i, W, w, t, r)$
$s_{i+1} \leftarrow s_i \cdot r$
$s'_{i+1} \leftarrow s'_i \cdot r^{-1}$
Return $((T, w), \Lambda_i, \Lambda'_i)$

sim $VRFYSIM((T, w), m, l_1, l_2)$:
$W \leftarrow H(m, w)$
$r \xleftarrow{\$} \mathcal{G}_1$
$t_i \leftarrow e(s_i, W)$
$\Lambda_i \leftarrow l_1(s_i)$
$\Lambda'_i \leftarrow l_2(s'_i, W, w, t, r)$
$s_{i+1} \leftarrow s_i \cdot r$
$s'_{i+1} \leftarrow s'_i \cdot r^{-1}$
$b \leftarrow VRFY((T, w), m)$
Return $(b, \Lambda_i, \Lambda'_i)$

Fig. 12. Constructing a EUF-CMA adversary against $\mathcal{M}$ using an adversary against G_2