Efficient Leakage-Resilient MACs without Idealized Assumptions

Francesco Berti[1,2], Chun Guo[3,4,5], Thomas Peters[1], François-Xavier Standaert[1]

[1] UCLouvain, ICTEAM/ELEN/Crypto Group
[2] TU Darmstadt, Germany, CAC - Applied Cryptography
[3] School of Cyber Science and Technology, Shandong University, Qingdao, Shandong, 266237, China
[4] Key Laboratory of Cryptologic Technology and Information Security of Ministry of Education, Shandong University, Qingdao, Shandong, 266237, China
[5] State Key Laboratory of Information Security (Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093)

Abstract. The security proofs of leakage-resilient MACs based on symmetric building blocks currently rely on idealized assumptions that hardly translate into interpretable guidelines for the cryptographic engineers implementing these schemes. In this paper, we first present a leakage-resilient MAC that is both efficient and secure under standard and easily interpretable black box and physical assumptions. It only requires a collision resistant hash function and a single call per message authentication to a Tweakable Block Cipher (TBC) that is unpredictable with leakage. This construction leverages two design twists: large tweaks for the TBC and a verification process that checks the inverse TBC against a constant. It enjoys beyond birthday security bounds. We then discuss the cost of getting rid of these design twists. We show that security can be proven without them as well. Yet, a construction without large tweaks requires stronger (non idealized) assumptions and may incur performance overheads if specialized TBCs with large tweaks can be exploited, and a construction without twisted verification requires even stronger assumptions (still non idealized) and leads to more involved bounds. The combination of these results makes a case for our first pragmatic construction and suggests the design of TBCs with large tweaks and good properties for side-channel countermeasures as an interesting challenge.

1 Introduction

Ever since its introduction by Dziembowski and Pietrzak [20], leakage-resilient cryptography has been characterized by a quest towards the best tradeoff between weak physical assumptions and efficient cryptographic constructions. Finding good abstractions to limit the informativeness of the leakage function, that can be fulfilled by hardware engineers while also enabling sound security proofs, is a typical example of this challenge. Current assumptions range from various types of "bounded leakage", as comprehensively discussed in [21], to simpler solutions leveraging the scarce use of "strongly protected components", modeled

as leak-free in [33]. Unsurprisingly, the most efficient (symmetric) constructions in the literature leverage such strong (idealized) assumptions [6].

While avoiding idealized assumptions is of general interest in cryptography, it is even more desirable in leakage-resilient cryptography, since perfectly ensuring a physical assumption may not be possible, or only at prohibitive cost. For example, instantiating a (128-bit) leak-free component would require masking it at very high security orders, leading to significant performance overheads [22], while security against $< 2^{80}$ measurements may be sufficient for a majority of applications. So despite proofs relying on a leak-free component are a useful guide towards efficient constructions with good leakage properties, interpreting their security bounds in terms of concrete requirements for cryptographic engineers (e.g., in terms of a level of protection to reach in practice) remains difficult.

Given this state-of-the-art, the design of leakage-resilient MACs appears as a first natural target. As put forward by Micali and Reyzin, ensuring unpredictability in the presence of leakage is significantly easier than ensuring indistinguishability in the presence of leakage [32]. This observation led the authors of [10] to propose authenticated encryption schemes for which the integrity holds even if the vast majority of its (ephemeral) secrets are leaked in full to the adversary: a model that we next denote as the "unbounded leakage model". Yet, these authenticated encryption schemes (and follow ups next listed as related works) still rely on the scarce utilization of a leak-free component.

An intermediate step towards getting rid of the leak-free component has been made by Berti et al. [8]: it shows that it is possible to replace this leak-free component by a (tweakable) block cipher ensuring "strong unpredictability with leakage". The idea of basing MAC security on unpredictable ciphers is not new. To the best of our knowledge, it dates back to [1] and has been revisited in [18,36,19]. Its leakage generalization is specially appealing since such a game-based definition can then be verified/falsified by evaluation laboratories. Yet, the results in [8] still rely on a random oracle assumption, which seems mostly due to the difficult interaction between an unpredictable (tweakable) block cipher and the hash part of their constructions. As for the leak-free component, such an idealized assumption is in general undesirable, and possibly even more when leakage comes into play. So the main question we tackle in this work is: *can we design leakage-resilient MACs without idealized assumptions (i.e., no leak-free component nor random oracles)?* We answer it positively by exhibiting efficient constructions for which the leakage security holds in the unbounded leakage model, only requiring strong unpredictability with leakage for their tweakable block cipher and (more or less) standard properties for their hash function.

More precisely, our contributions are threefold:

We first propose a pragmatic construction (next denoted as LR-MAC1) that takes advantage of simple design twists. The starting observation for this purpose is that the HTBC construction in [8] performs the verification by checking whether an inverted Tweakable Block Cipher (TBC) matches the output of a hash function. By changing this verification and checking whether the inverted TBC equals a constant (e.g., zero) value, we avoid the difficult interaction be-

tween the hash function and the TBC that has led this previous work to rely on a random oracle assumption. We then show that if the construction is instantiated with a TBC having $2n$-bit tweaks, it provides tight and beyond-birthday security bounds, under standard assumptions (namely, strong unpredictability with leakage for the TBC and collision-resistance for the hash function). While this construction positively answers our question, it still relies on two design twists, namely a TBC with $2n$-bit tweaks and the introduction of a constant value in the verification process. So we complement our first pragmatic constructions by two other designs aiming to clarify whether these twists are necessary.

Our second design (next denoted as LR-MAC2) is a variant of LR-MAC1 that only relies on TBCs having n-bit tweaks. We show that it is possible to maintain leakage security without idealized assumptions with this simpler building block. However, designs reaching this goal currently need to rely on two calls to the TBC, which is in general more expensive than a single call to a TBC with $2n$-bit tweaks if a specialized TBC can be used [27]. Besides, the best construction we reach also requires a less standard (yet non idealized) assumption on its hash function (namely, collision-resistance for one half of its output). The latter may therefore require more rounds in the instances of hash functions it uses.

Eventually, we revisit the leakage security of the HTBC design (which does not use a constant value in its verification), analyzed in [8] under a random oracle assumption. We show that such a construction can be proven secure without this idealized assumption, but at the cost of more involved bounds and stronger (still non idealized) assumptions for the hash function than LR-MAC2.

Overall, the combination of our observations regarding LR-MAC2 and HTBC strengthen the interest of the pragmatic LR-MAC1 solution, and suggest the design of TBCs with $2n$-bit tweaks and good properties to be protected via masking as natural design targets for leakage-resilient modes of operation.

Related works. Leakage-resilient MACs have first been proposed by Hazay et al. [25] and Martin et al. [31], but imply higher performance overheads than the symmetric constructions we consider in this work. The first leakage-resilient MACs based on symmetric building blocks were proposed in [34,33]. Several (TBC-based or permutation-based) leakage-resilient authenticated encryption schemes have been proposed and embed a leakage-resilient MAC for their integrity guarantees, e.g., [14,16,9,15,24,23,15,30]: they all rely on idealized assumptions to some extent (see [6] for an overview). The specific issue of comparing values (e.g., tags) in a leakage-resilient manner is discussed in [11,17].

2 Background

Notations. With $\{0,1\}^n$ (resp., $\{0,1\}^*$), we denote the set of all strings of length n (resp., all finite-length strings). With $|\mathcal{X}|$, we denote the size of the set $\mathcal{X}$. In some arguments, we denote by n the security parameter, $\mathsf{negl}(n)$ negligible functions and $\mathsf{poly}(n)$/$\mathsf{superpoly}(n)$ polynomial/super-polynomial functions.

2.1 Primitives: hash functions and TBC

Our schemes use hash functions and TBCs. For hash functions, the minimum property we require is collision-resistance, which we recall next:

Definition 1. *Let* $\mathsf{H} : \mathcal{HK} \times \{0,1\}^* \to \mathcal{X}$ *be a hash function.* H *is* (t, ϵ)-collision resistant (CR) *if for every* t*-bounded adversary* A*, the probability that* $\mathsf{A}(s)$ *outputs a pair of distinct inputs* $(m^0, m^1) \in (\{0,1\}^*)^2$*, such that* $\mathsf{H}_s(m^0) = \mathsf{H}_s(m^1)$ *and* $m^0 \neq m^1$*, is bounded by* ϵ*, with* $s \xleftarrow{\$} \mathcal{HK}$ *picked uniformly at random with:*

$$\Pr[s \xleftarrow{\$} \mathcal{HK}, \mathsf{A}(s) \Rightarrow (m^0, m^1) \in (\{0,1\}^*)^2 \text{ s.t. } m^0 \neq m^1, \mathsf{H}_s(m^0) = \mathsf{H}_s(m^1)] \leq \epsilon.$$

We will sometimes require range-oriented preimage resistant hash functions:

Definition 2. *Let* $\mathsf{H} : \mathcal{HK} \times \{0,1\}^* \to \mathcal{X}$ *be a hash function.* H *is* (t, ϵ)-range-oriented preimage resistant (rpre) *if, for every* t*-bounded adversary* A*:*

$$\Pr[s \xleftarrow{\$} \mathcal{HK}, y \xleftarrow{\$} \mathcal{X}, \mathsf{A}(s, y) \Rightarrow m \in \{0,1\}^* \text{ s.t. } \mathsf{H}_s(m) = y] \leq \epsilon.$$

Properties of rpre hashing. Following [2], a hash function can be characterized by a table of preimage computing probabilities.

Definition 3. *Let* A *be an adversary against the preimage resistance of a hash function, then the associated success probability matrix* $M \in [0,1]^{|\mathcal{HK}| \times |\mathcal{X}|}$ *is:*

$$M_{s,y} = \Pr_{\mathsf{A}(s,y) \Rightarrow m}\big[\mathsf{H}_s(m) = y\big],$$

defined entry-wise where $s \in \mathcal{HK}$ *and* $y \in \mathcal{X}$*, and where the probability is taken over the random coins of* A*.*

Consider a hash function $\mathsf{H} : \mathcal{HK} \times \{0,1\}^* \to \mathcal{X}$ and an arbitrary adversary A. The intuition is that the preimage for some "weak" points in $\mathcal{X}$ may be easily computable. But if H is range-oriented preimage resistant, then the total number of such "weak" points should be limited. To formalize this idea, for any $s \in \mathcal{HK}$, we define a set for such "weak" points as:

$$\mathcal{WP}(s, \mathsf{A}) = \Big\{ y \in \mathcal{X} : \Pr_{\mathsf{A}(s,y) \Rightarrow m}\Big[\mathsf{H}_s(m) = y\Big] = \mathsf{poly}(n)^{-1} \Big\}. \tag{1}$$

With the above definition, we are able to establish the following claim.

Lemma 1. *If* H *is range-oriented preimage resistant, then with high probability, the size of the set* $\mathcal{WP}(s, \mathsf{A})$ *of "weak" images is polynomial with:*

$$\Pr\big[s \xleftarrow{\$} \mathcal{HK} : |\mathcal{WP}(s, \mathsf{A})| = \mathsf{superpoly}(n)\big] = \mathsf{negl}(n).$$

Proof. To see this, we expand the expression as follows:

$$\begin{aligned}
& \Pr[s \xleftarrow{\$} \mathcal{HK}, y \xleftarrow{\$} \mathcal{X}, \mathsf{A}(s,y) \Rightarrow m \in \{0,1\}^* \text{ s.t. } \mathsf{H}_s(m) = y], \\
= & \sum_{s \in \mathcal{HK}} \frac{1}{|\mathcal{HK}|} \sum_{y \in \mathcal{X}} \frac{1}{|\mathcal{X}|} \Pr[\mathsf{A}(s,y) \Rightarrow m \in \{0,1\}^* \text{ s.t. } \mathsf{H}_s(m) = y], \\
\geq & \sum_{s \in \mathcal{HK}} \frac{1}{|\mathcal{HK}|} \sum_{y \in \mathcal{WP}(s,\mathsf{A})} \frac{1}{|\mathcal{X}|} \cdot \mathsf{superpoly}(n), \\
\geq & \sum_{s \in \mathcal{HK},\ |\mathcal{WP}(s,\mathsf{A})| = \mathsf{superpoly}(n)} \frac{1}{|\mathcal{HK}|} \frac{|\mathcal{WP}(s,\mathsf{A})|}{|\mathcal{X}|} \cdot \mathsf{superpoly}(n), \\
\geq & \Pr\left[s \xleftarrow{\$} \mathcal{HK} : |\mathcal{WP}(s,\mathsf{A})| = \mathsf{superpoly}(n)\right] \cdot \frac{\mathsf{superpoly}(n)}{|\mathcal{X}|}. \qquad (2)
\end{aligned}$$

By this, if $\Pr\left[s \xleftarrow{\$} \mathcal{HK} : |\mathcal{WP}(s,\mathsf{A})| = \mathsf{superpoly}(n)\right]$ is not $\mathsf{negl}(n)$, then Eq. (2) cannot be negligible, contradicting the range-oriented preimage resistance assumption. The claim thus follows. □

Thanks to Lemma 1, it *may* be feasible to decide the set $\mathcal{WP}(s,\mathsf{A})$ for any adversary A and any seed s. In Section 5, Theorem 3, this result will allow us to assume that the set $\mathcal{WP}(s,\mathsf{A})$ for the hash function used in our design is computable in Probabilistic Polynomial-Time (PPT).

For TBC, the minimum property we require is strong pseudorandomness:

Definition 4 (stPRP). *A tweakable block cipher* $\mathsf{F} : \mathcal{K} \times \mathcal{TW} \times \{0,1\}^n \to \{0,1\}^n$ *is a* (q,t,ϵ)-strong tweakable pseudorandom permutation (stPRP) *if* $\forall (k,tw) \in \mathcal{K} \times \mathcal{TW}, \mathsf{F}_k^{tw} : \{0,1\}^n \to \{0,1\}^n$ *is a permutation and if for every* (q,t)*-adversary* A*, the advantage :*

$$\mathsf{Adv}_{\mathsf{F}}^{\mathsf{stPRP}}(\mathsf{A}) := \left| \Pr\left[\mathsf{A}^{\mathsf{F}_k(\cdot,\cdot),\mathsf{F}_k^{-1}(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathsf{A}^{\mathsf{f}(\cdot,\cdot),\mathsf{f}^{-1}(\cdot,\cdot)} \Rightarrow 1\right] \right|,$$

is upper bounded by ϵ*, where* k *and* f *are chosen uniformly at random from their domains, namely* $\mathcal{K}$ *and the space* $\mathcal{TPERM}(\mathcal{TW}, \{0,1\}^n)$ *of tweakable permutations (i.e., the space of functions* $\mathsf{f} : \mathcal{TW} \times \{0,1\}^n \to \{0,1\}^n$ *s.t.* $\forall tw \in \mathcal{TW}, \mathsf{f}(tw,\cdot) : \{0,1\}^n \to \{0,1\}^n$ *is a permutation). The adversary can do at most* q' *queries to the first oracle and* $q - q'$ *queries to the second one for any* $q' \leq q$*.*

For simplicity, an (n,n,n)-TBC is a TBC with $\mathcal{K} = \mathcal{TW} = \{0,1\}^n$.

2.2 MAC (Message Authentication Code)

We recall the definition of Message Authentication Code (MAC) as follows:

Definition 5. *A* MAC *is a triple* $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ *where:*

Gen. *The key-generation algorithm* Gen *picks a key in the keyspace* $\mathcal{K}$*.*

Mac. *The tag-generation algorithm* Mac *takes as input a couple* $(k, m) \in \mathcal{K} \times \{0,1\}^*$ *and outputs a tag* $\tau \leftarrow \mathsf{Mac}_k(m)$ *from the tag space* $\mathcal{TAG}$.

Vrfy. *The verification algorithm* Vrfy *takes as input a triple* (k, m, τ) *in* $\mathcal{K} \times \{0,1\}^* \times \mathcal{TAG}$ *and outputs either "*$\top$*"* ("accept") *or "*$\bot$*"* ("reject").

We require correctness: $\forall (k, m) \in \mathcal{K} \times \{0,1\}^*,\ \mathsf{Vrfy}(k, m, \mathsf{Mac}(k, m)) = \top$.

Since we will only consider the security for MACs in the presence of leakage, we omit the security definition in the black-box model (that is, when there is no leakage). This definition can be found in many works, for example [29].

2.3 Leakage models and security definitions with leakage

Notations. When an adversary has access not only to the outputs of an oracle but also to its leakage, we denote it with A^{OL}. In this case, queries to the oracle OL on input x are answered with $y = \mathsf{O}(x)$ and the leakage $\mathsf{l}_{\mathsf{O}} := \mathsf{L}_{\mathsf{O}}(x)$. If the oracle is keyed with the key k, we write the leakage function as $\mathsf{L}_{\mathsf{O}}(x; k)$. Finding good abstractions to restrict L_{O} is in general a hard problem [21].[6]

Strong unforgeability with leakage (sUF–L2). We start by introducing the security for MACs in the presence of leakage. We want that it is hard to forge valid tags even having access to the leakage of the tag-generation and the verification algorithms (that is, finding a fresh and valid couple message tag (m, τ) such that $\mathsf{Vrfy}_k(m, \tau) = \top$ should be hard). We use the sUF–L2 definition of Berti et al. [8] for this purpose, which we recall next:

Definition 6 (sUF–L2). *A* $\mathsf{MAC} = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ *with tag-generation leakage function* L_M *and verification leakage function* L_V *is* $(q_L, q_M, q_V, t, \epsilon)$-strongly existentially unforgeable against chosen message and verification attacks with leakage in the tag-generation and the verification (sUF–L2) *if for all* (q_L, q_M, q_V, t)-*adversaries* A^{L}*, we have:*

$$\Pr\left[\mathsf{FORGEL2}^{\mathsf{suf\text{-}vcma\text{-}L2}}_{\mathsf{MAC},\mathsf{L}_M,\mathsf{L}_V,\mathsf{A}} \Rightarrow 1\right] \leq \epsilon,$$

where the $\mathsf{FORGEL2}^{\mathsf{suf\text{-}vcma\text{-}L2}}$ *experiment is defined in Table 1.*

For simplicity, we consider the verification query induced by the final output of the adversary as the $(q_v + 1)th$ verification query.

[6] Adversaries are sometimes allowed to "model" the leakage. For this purpose, we grant them access to the oracle L. This oracle is peculiar since it allows the adversary to make queries not only on inputs x but also of keys k' of its choice.

The $\mathsf{FORGEL2}^{\mathsf{suf\text{-}vcma\text{-}L2}}_{\mathsf{MAC},\mathsf{L}_M,\mathsf{L}_V,\mathsf{A}^{\mathsf{L}}}$ experiment	
Initialization: $k \leftarrow \mathsf{Gen}$ $\mathcal{S} \leftarrow \emptyset$ Finalization: $(m,\tau) \leftarrow \mathsf{A}^{\mathsf{L},\mathsf{MacL}_k(\cdot),\mathsf{VrfyL}_k(\cdot,\cdot)}$ If $(m,\tau) \in \mathcal{S}$ or $\perp = \mathsf{Vrfy}_k(m,\tau)$ Return 0 Return 1	Oracle $\mathsf{MacL}_k(m)$: $\tau = \mathsf{Mac}_k(m)$ $\mathcal{S} \leftarrow \mathcal{S} \cup \{(m,\tau)\}$ Return $(\tau, \mathsf{L}_M(m;k))$ Oracle $\mathsf{VrfyL}_k(m,\tau)$: Return $(\mathsf{Vrfy}_k(m,\tau), \mathsf{L}_V(m,\tau;k))$

Table 1: The $\mathsf{FORGEL2}^{\mathsf{suf\text{-}vcma\text{-}L2}}$ experiment.

The unbounded leakage model. We next need to specify the functions we will use for L_M and L_V. Based on the aforementioned difficulty to restrict the leakage in a meaningful manner, we first follow the observation made in [33] that it is possible to implement leakage-resilient cryptographic functionalities so that the execution of most underlying building blocks can leak in an unrestricted manner. The resulting "leveled implementations" only require a few calls to a strongly protected component (frequently modeled as leak-free) to ensure the desired security property (here, sUF–L2). So we will next consider the unbounded leakage model, where the leakage function yields all the internal states produced during each execution of the scheme under investigation, at the exclusion of the strongly protected components that are used to manipulate long-term secrets. More precisely, in the unbounded leakage model:

- Unprotected building blocks leak their inputs, outputs and keys in full;
- Building blocks with strongly protected implementation leak their inputs and outputs in full and their key only leaks in a restricted manner.

In practice, the only strongly protected component we will use in our construction is a TBC, and we next specify how its leakage will be restricted by asking implementers to ensure its strong unpredictability with leakage.

Strong unpredictability with leakage (sUP–L2). Unpredictability is among the simplest requirements for (tweakable) block ciphers. As mentioned in introduction, its application in leakage-resilient cryptography is appealing since it corresponds to a game-based definition that can directly be tested by an evaluation laboratory. We next give its definition which formalizes that it should be hard for an adversary to find a triple (tw, x, y) which is fresh and valid (i.e., $y = \mathsf{F}_k^{tw}(x)$) even if the leakage function $\mathsf{L} = (\mathsf{L}_{\mathsf{Eval}}, \mathsf{L}_{\mathsf{Inv}})$ associated to an implementation of the TBC can be queried, with $\mathsf{L}_{\mathsf{Eval}}(tw, x; k)$ (resp., $\mathsf{L}_{\mathsf{Inv}}(tw, z; k)$) the leakage resulting from the computation of $\mathsf{F}_k(tw, x)$ (resp., $\mathsf{F}_k^{-1}(tw, z)$).

Definition 7 (sUP–L2). *A tweakable block cipher* $\mathsf{F} : \mathcal{K} \times \mathcal{TW} \times \{0,1\}^n \rightarrow \{0,1\}^n$ *with leakage function pair* $\mathsf{L} = (\mathsf{L}_{\mathsf{Eval}}, \mathsf{L}_{\mathsf{Inv}})$ *is* $(q_{\mathsf{L}}, q_E, q_I, t, \epsilon)$-strongly un-

predictable with leakage in evaluation and inversion (sUP–L2), *or* $(q_{\mathsf{L}}, q_E, q_I, t, \epsilon)$-sUP–L2*, if for any* $(q_{\mathsf{L}}, q_E, q_I, t)$*-adversary* A*, we have:*

$$\Pr[\mathsf{sUP\text{–}L2}_{\mathsf{A},\mathsf{F},\mathsf{L}} \Rightarrow 1] \leq \epsilon,$$

where the sUP–L2 *experiment is defined in Table 2, and* A^{L} *makes at most* q_{L} *(offline) queries to* L *(used to model the leakage, cf. Footnote 6).*

When the adversary has access only to the l_{e} oracle, the TBC is *unpredictable with leakage* (as previously defined by Dodis and Steinberger [18]).[7]

The $\mathsf{sUP\text{–}L2}_{\mathsf{A},\mathsf{F},\mathsf{L}}$ experiment.	

Initialization:
$k \xleftarrow{\$} \mathcal{K}$
$\mathcal{L} \leftarrow \emptyset$
 	
Finalization:
$(x, tw, z) \leftarrow \mathsf{A}^{\mathsf{L},\mathsf{LEval}(\cdot,\cdot),\mathsf{LInv}(\cdot,\cdot)}$
If $(x, tw, z) \in \mathcal{L}$
  Return 0
If $z = \mathsf{F}_k(tw, x)$
  Return 1
Return 0
	Oracle $\mathsf{LEval}(tw, x)$:
	$z = \mathsf{F}_k(tw, x)$
	$\mathsf{l}_{\mathsf{e}} = \mathsf{L}_{\mathsf{Eval}}(tw, x; k)$
	$\mathcal{L} \leftarrow \mathcal{L} \cup \{(x, tw, z)\}$
	Return $(z, \mathsf{l}_{\mathsf{e}})$
	
	Oracle $\mathsf{LInv}(tw, z)$:
	$x = \mathsf{F}_k^{-1}(tw, z)$
	$\mathsf{l}_{\mathsf{i}} = \mathsf{L}_{\mathsf{Inv}}(tw, z; k)$
	$\mathcal{L} \leftarrow \mathcal{L} \cup \{(x, tw, z)\}$
	Return $(x, \mathsf{l}_{\mathsf{i}})$

Table 2: Strong unpredictability with leakage in evaluation and inversion.

Concretely, this assumption is significantly closer to practice than idealized ones. In particular, breaking it requires one to fully compute the value of the block cipher on a new point with non-trivial probability [18]. Therefore, we expect that satisfying unpredictability with leakage mainly requires protecting the long-term key, and that testing block cipher implementations against side-channel key recovery attacks, which is the current focus of evaluation laboratories [4], should give a good indication of their unpredictability with leakage.

3 Design and analysis of LR-MAC1

In this section, we present LR-MAC1, which is sUF–L2 in the unbounded leakage model, assuming a collision-resistant hash function and a sUP–L2 TBC.

[7] Degabriele et al. give an alternative definition in [14]. The main difference is that the set of inputs ($\mathcal{S}$ in their definition, $\mathcal{L}$ in ours) is not increased for inputs X for which only the leakage is observed, while we always give access to both the primitive's output and their leakage. Hence, this definition cannot be satisfied in the unbounded leakage model as we aim, since the adversary can then get a valid tag in full. (Their motivation was also different from ours and specially tailored for analyzing constructions leveraging leakage-resilient PRFs).

The main design idea used by LR-MAC1 is to avoid that a value obtained by leakage in verification may be used for a future forgery. This happened in the HTBC leakage-resilient MAC of [8] because the output of an (inverse) TBC is compared with a hash value $h = \mathsf{H}(m)$. As illustrated in Figure 1, LR-MAC1 prevents this by doing the verification check with a fixed value that does not depend on the message m. In this figure, strongly protected components are in gray, unprotected components are in white, inputs and outputs of the scheme (together with values that can be computed publicly from them) are in green, intermediate values that leak (in an unbounded manner in our case) are in orange and long-term secrets manipulated by strongly protected components are in red.

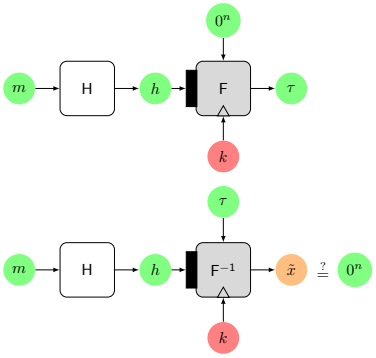

Fig. 1: LR-MAC1.

Concretely, the hash of the message is used only as a tweak of the TBC, while the input of the TBC is always a fixed value (i.e., 0^n). In decryption, we check if the inverse of the tag $\tilde{x} := \mathsf{F}_k^{-1}(\tau)$ is 0^n. This way, even an unbounded leakage during an invalid verification cannot lead to efficient forgeries since $\tilde{x}$ cannot be reused (contrary to what happens for HTBC). Moreover, since the hash of the message h is used only as a tweak, there is no target to find a preimage (without producing a collision). As a result, a verification query $\mathsf{Vrfy}_k(m, \tau) \rightarrow \top$ immediately results in a forgery $\big(\mathsf{H}_s(m), 0^n, \tau\big)$, which does not rely on the distribution of $\mathsf{H}_s(m)$. This eliminates the necessity of the random oracle in our proofs. The detailed specifications of LR-MAC1 can be found in Algorithm 1.

3.1 sUF–L2-security of LR-MAC1

We now prove that LR-MAC1 is sUF–L2 in the unbounded leakage model.

Unbounded leakage specification. Before assessing the sUF–L2 of LR-MAC1, we have to define the leakage that an adversary can collect:

Algorithm 1 The LR-MAC1 algorithm.

It uses a strongly protected TBC $\mathsf{F} : \mathcal{K} \times \mathcal{TW} \times \{0,1\}^n \to \{0,1\}^n$ and a hash function $\mathsf{H} : \mathcal{HK} \times \{0,1\}^n \to \mathcal{TW}$.

- Gen
  - $k \xleftarrow{\$} \mathcal{K}$
  - $s \xleftarrow{\$} \mathcal{HK}$
- $\mathsf{Mac}_k(m)$:
  - $h = \mathsf{H}_s(m)$
  - $\tau = \mathsf{F}_k^h(0^n)$
  - Return τ
- $\mathsf{Vrfy}_k(m, \tau)$:
  - $h = \mathsf{H}_s(m)$
  - $\tilde{x} = \mathsf{F}_k^{h,-1}(\tau)$
  - If $\tilde{x} == 0^n$ Return $\top$
  - Else Return $\bot$

- $\mathsf{L}_M(m; k)$ returns $\mathsf{LEval}(0^n, h; k)$ with $h = \mathsf{H}_s(m)$.[8]
- $\mathsf{L}_V(m, \tau; k)$ returns $\tilde{x}$ and $\mathsf{LInv}(\tau, h; k)$ with $h = \mathsf{H}_s(m)$.

Our main result for LR-MAC1 is then formalized by the following theorem:

Theorem 1. *Let* H *be a* $(t_1, \epsilon_{\mathsf{CR}})$*-collision resistant hash function. Let* F *be a* $(q_{\mathsf{L}}, q_M, q_V, t_2, \epsilon_{\mathsf{sUP\text{-}L2}})$-sUP–L2 TBC*. Then, LR-MAC1 is* (q_M, q_V, t, ϵ)-sUF–L2*-secure in the unbounded leakage model with:*

$$\epsilon \leq \epsilon_{\mathsf{CR}} + (q_V + 1)\ \epsilon_{\mathsf{sUP\text{-}L2}},$$

with $t_1 = t + (q_M + q_V + 1)t_{\mathsf{H}} + (q_M + q_V)(t_{\mathsf{F}} + t_{\mathsf{L(F)}})$ *and* $t_2 = t + (q_M + q_V + 1)t_{\mathsf{H}}$*, where* t_{H} *is the time needed to execute once the hash function* H*,* t_{F} *is the time needed to execute once the* TBC F *and* $t_{\mathsf{L(F)}}$ *is the time needed to collect the leakage of one execution of the* TBC F*.*

Proof. We use a sequence of games. To make the proof simpler, we give only a sketch of the adversaries used. The details can be found in Appendix A.

Game 0. Let Game 0 be the sUF–L2 game where the (q_M, q_V, t)-adversary A^{L} tries to produce a forgery when she plays against LR-MAC1. Let E_0 be the event that the adversary wins the game (i.e., the output of the game is 1).

Game 1. Game 1 is Game 0 except that we abort if there is a collision in the hash function. Let E_1 be the event that the adversary wins the game.

[8] h can be computed by the adversary since the key s of the hash function is public.

Transition between Game 1 and Game 2. Clearly, Game 0 and Game 1 are identical if the following event HC (*Hash Collision*) does not happen:

$HC := \{\exists i, j \in \{1, ..., q_M\} \cup \{1, ..., q_V+1\}$ with $i \overset{\%}{=} j$ s.t. $h^i = h^j$ and $m^i \neq m^j\}$.[9]

To compute this event, we build a t_1-CR-adversary B.

The t_1-CR-adversary B is an adversary playing against the hash function, based on A, which does the hash queries induced by the queries of A and then correctly computes the answer for A. At the end of the game, B sees if her hash queries have produced a collision. If it is the case, she outputs it. This adversary needs time $t + (q_M + q_V + 1)t_{\mathsf{H}} + (q_M + q_V)(t_{\mathsf{F}} + t_{\mathsf{L(F)}})$.

Bounding $|\Pr[E_0] - \Pr[E_1]|$. Observe that if event HC happens, B wins. Note that B simulates correctly Game 0 for A^{L}. Since B is a t_1-adversary and H is a $(t_1, \epsilon_{\mathsf{CR}})$-collision resistant hash function, we can bound:

$$|\Pr[E_0] - \Pr[E_1]| \leq \epsilon_{\mathsf{CR}}.$$

Game 2. Game 2 is Game 1 except that we abort if one verification query (or the final one) made by A is fresh and valid. Let E_2 be the event that the adversary wins the game (i.e., the output of the game is 1).

Transition between Game 1 ad Game 2. We build a sequence of $q_V + 2$ games: Game 1^0,...,Game 1^{q_V+1} as follows.

Game 1^j. Game 1^j is Game 1 where we abort if one of the first j verification queries is fresh and valid. Thus, Game 1^0 is Game 1 while Game 1^{q_V+1} is Game 2. Let E_1^j be the event that the adversary wins the game.

Transition between Game 1^j and Game 1^{j+1}. Clearly, Game 1^j and Game 1^{j+1} are identical if the $j+1$-th verification query is either invalid or not fresh. If this query is fresh and valid, we say that event GT^j (Good Tag) happens. In order to bound the probability that event GT^j happens we build a $(q_{\mathsf{L}}, q_{\mathsf{Eval}}, q_{\mathsf{Inv}}, t_2)$-$\mathsf{sUP\text{-}L2}$-adversary C^j against F as follows.

The $(q_{\mathsf{L}}, q_{\mathsf{Eval}}, q_{\mathsf{Inv}}, t_2)$-$\mathsf{sUP\text{-}L2}$-adversary C^j has to find a valid triple (x, tw, y) for F_k which is fresh and valid. She simulates Game 1^j for A until the $j+1$th verification query. Then, when A asks his $j+1$-th verification query on input $(m^{q_V+1}, \tau^{q_V+1})$, C^j computes (1) $h^{j+1} = \mathsf{H}_s(m^{j+1})$, and (2) she outputs $(0^n, h^{j+1}, \tau^{j+1})$. This takes time t_{H}. In total, C^i does at most q_{L} query to L, q_M to LEval and $j \leq q_v$ to LInv. She needs time at most $t + (q_M + j + 1)t_{\mathsf{H}} \leq t_2$.

[9] $i \overset{\%}{=} j$ means that if i comes from a tag-generation query and j from a verification query, or vice-versa, then they are considered differently.

Bounding $\big|\Pr[E_1^j]-\Pr[E_1^{j+1}]\big|$**.** Note that C^i simulates correctly Game 1^j for A^{L}. Since C^j is a $(q_{\mathsf{L}}, q_M, q_V, t_2)$-adversary and F is $(q_{\mathsf{L}}, q_M, q_V, t_2, \epsilon_{\mathsf{sUP\text{-}L2}})$-sUP–L2, we can bound:

$$|\Pr[E_1^j] - \Pr[E_1^{j+1}]| \leq \Pr[GT^{j+1}] \leq \epsilon_{\mathsf{sUP\text{-}L2}}.$$

Bounding $\big|\Pr[E_1] - \Pr[E_2]\big|$**.** Iterating we can bound:

$$\big|\Pr[E_1] - \Pr[E_2]\big| \leq \sum_{j=0}^{q_V} \big|\Pr[E_1^j] - \Pr[E_1^{j+1}]\big| \leq (q_V+1)\epsilon_{\mathsf{sUP\text{-}L2}}.$$

Concluding the proof. Since the probability of E_2 is 0, we can conclude the proof putting everything together:

$$\Pr[E_0] = \sum_{i=0}^{1} \big|\Pr[E_i] - \Pr[E_{i+1}]\big| + \Pr[E_2] = \epsilon_{\mathsf{CR}} + (q_V+1)\epsilon_{\mathsf{sUP\text{-}L2}} + 0.$$

3.2 Instantiation and concrete security

Built upon a "good enough" hash function, the term ϵ_{CR} is expected to be in $O(t_1^2/2^{|\mathcal{TW}|})$. On the other hand, the term $\epsilon_{\mathsf{sUP\text{-}L2}}$ depends on the concrete side-channel strength of the TBC implementation. Since we have a birthday bound only on the size of the tweak space $\mathcal{TW}$ and not on the block space $\mathcal{B} = \{0,1\}^n$, a natural idea is to use a TBC with the tweak space bigger than the block space. For example, a good instantiation would be to use SHA256 as hash function and Deoxys-384 (which has a 128-bit block and 256-bit tweak [28]) as TBC. In this case, finding a collision requires the computation of around 2^{128} hash functions, and therefore, the $(q_V+1)\epsilon_{\mathsf{sUP\text{-}L2}}$ term dominates the bound.[10]

4 Design and analysis of LR-MAC2

In this section, we question the possibility to design a leakage-resilient MAC with a (more standard) TBC design having only n-bit tweaks rather than $2n$ for LR-MAC1. We show that this is achievable at the cost of a second call per message authentication to the TBC and by requiring an additional property of the hash function that we develop after the description of the scheme. Our security analysis also holds without idealized assumption.

4.1 Description of LR-MAC2

To get high security we start again from $\mathsf{H}_s(m) = u\|v$, where $u, v \in \{0,1\}^n$, as illustrated in Figure 2. As in HTBC, we first compute $\mathsf{F}_k^v(u)$ with $k = k_1$ to get a

[10] In the black box setting (i.e., without leakage), the security of this construction is beyond birthday since $(q_V+1)\ \epsilon_{\mathsf{sUP\text{-}L2}} \leq \epsilon_{\mathsf{sPRP}} + \frac{q_V+1}{2^{128}-q_M-q_V}$, which is optimal.

value dependent on both n-bit strings u and v. To verify the validity of a tag τ for message m, we know that we should rely on a TBC inversion call to avoid leaking useful information about the good tag if (m, τ) is actually an invalid message-tag pair. However, we also have to circumvent the difficulty encountered when dealing with the verification of HTBC tags, as discussed in the introduction. That means that we should use our second TBC call in such a way that the equality check must not directly involve either u or v (so, a half hash value) and an output of the TBC (thus requiring to rely on a backwards computation). As a result, we need a serial evaluation of the TBC calls to compute a tag, e.g., as $\tau = \left(\mathsf{F}^v_{k_2} \circ \mathsf{F}^v_{k_1}\right)(u)$, with an equality check performed in the middle to compare $\mathsf{F}^v_{k_1}(u)$ and $\mathsf{F}^{v,-1}_{k_2}(\tau')$ in the verification for a candidate tag τ'.

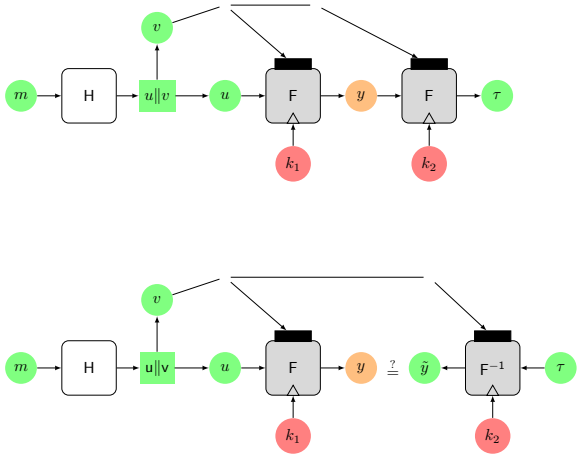

Fig. 2: LR-MAC2.

Up to switching the role between u and v, we now explain why $\mathsf{F}^v_{k_2} \circ \mathsf{F}^v_{k_1}$ is the good choice beyond the fact that using only a secret key primitive for the equality check allows us to learn what is computed and when, contrarily to the hash function. Since F_{k_2} and F_{k_1} are independent instances of the TBC, using the same tweak still enable us to use the output of one call against each other. The advantage of re-using the same tweak is twofold. First, we can easily partition all the queries with respect to v, the "lower half" of $\mathsf{H}_s(m) = u\|v$. This avoids considering many intermediate cases in the proof since with another computation of the form $\mathsf{F}^u_{k_2}$, with tweak u, we would have to deal with many "crossed" half hashes like $u_1\|v_1$, $u_1\|v_2$, $u_2\|v_1$... Second, given v, $\mathsf{F}^v_{k_2} \circ \mathsf{F}^v_{k_1}$ is a permutation between u and τ. We will see that this also simplifies the distribution of the potential forgeries, which in turn improves the security bound.

The complete specification of LR-MAC2 is given in Algorithm 2.

Algorithm 2 The LR-MAC2 algorithm.

It uses a strongly protected TBC $\mathsf{F} : \mathcal{K} \times \mathcal{TW} \times \{0,1\}^n \to \{0,1\}^n$ and a hash function $\mathsf{H} : \mathcal{HK} \times \{0,1\}^* \to \{0,1\}^{2n}$.

- Gen
  - $k = (k_1, k_2) \xleftarrow{\$} \mathcal{K}^2$
  - $s \xleftarrow{\$} \mathcal{HK}$
- $\mathsf{Mac}_k(m)$:
  - $u \| v \xleftarrow{n} \mathsf{H}_s(m)$
  - $y = \mathsf{F}^v_{k_1}(u)$
  - $\tau = \mathsf{F}^v_{k_2}(y)$
  - Return τ
- $\mathsf{Vrfy}_k(m, \tau)$:
  - $u \| v \xleftarrow{n} \mathsf{H}_s(m)$
  - $y = \mathsf{F}^v_{k_1}(u)$
  - $\tilde{y} = \mathsf{F}^{v,-1}_{k_2}(\tau)$
  - If $y == \tilde{y}$ Return $\top$
  - Else Return $\bot$

Single key version. For the sake of simplicity, we described LR-MAC2 with two uniform and independent keys. However, with some extra work we can rely on a single secret key k. The careful reader will have noted that we cannot derive k_1 and k_2 from F_k as the TBC is only unpredictable with leakage, and not pseudorandom with leakage which is a controversy assumption in the standard model. So the keys k_1 and k_2 will not have the right distribution to assume that F_{k_1} and F_{k_2} are themselves unpredictable with leakage. Nevertheless, the other known alternatives work. We can either assume that the TBC supports one more bit-tweak, i.e., tweaks of size $n+1$, or we can drop one bit of v in order to force in both cases the use of a domain separation bit in the tweaks between the two calls of the TBC. For completeness, we depict the construction where we compute $\tau = \mathsf{F}^{v\|1}_k \circ \mathsf{F}^{v\|0}_k(u)$ in Appendix B, Figure 4 and Algorithm 4 .

4.2 Half multi-collision resistance

It is natural to combine a double output length hash function $\mathsf{H}_s(m) = h = u\|v \in \{0,1\}^{2n}$ with an (n,n,n)-TBC F to target high MAC unforgeability. But since the hash value h is too long it cannot be used directly, and we split it into two n-bit pieces u and v to input $\mathsf{F}^v_k(u)$, for instance, where k is the n-bit key. The values u and v thus have distinct purposes in the design and it is equally natural to investigate the properties of both halves separately. At high level, this therefore requires that $\mathsf{H}_s = \mathsf{H}^1_s \| \mathsf{H}^2_s$ comes with additional guarantees for H^1_s and H^2_s, which can be seen as a strengthened security requirement for H_s.

General purpose standard hash functions should be as close as possible to public random functions. Since our goal is to avoid relying on idealized assumption, we will use the intuition that H^1_s and H^2_s should be asymptotically as good

as H_s. That is, even if in practice hash functions are built for a precise output-length (e.g., 256 bits), it just corresponds to asking that H^1_s and H^2_s are 128-bit output-length secure (enough) hash functions.

Obviously, we cannot simply concatenate any two 128-bit output collision resistant hash functions to build H_s and hope boosting the collision resistance of the resulting construction, say, up to 2^{96} queries or more. Still, $\mathsf{H}^1_s(m) = u$ and $\mathsf{H}^2_s(m) = v$ have to behave as securely as possible and we can expect them to be range-oriented/second/...preimage resistant and/or collision resistant up to $2^{64\theta}$ queries, for some $\theta \in (0, 1)$ as close as possible to 1.

Building some good hash $\mathsf{H}_s = \mathsf{H}^1_s \| \mathsf{H}^2_s$ enjoying additional "half-security" is out of the scope of this paper, and this additional requirement will be part of the discussion confirming the interest of the pragmatic LR-MAC1 construction at the end of the section. Yet, we see the independent security of H^1_s and H^2_s as a valuable asset and a natural target as many constructions might benefit from using the two halves u and v of h at different places in a cryptographic scheme, leading to a better overall security. We assume that it could be ensured by standard constructions, at the cost of slightly increased parameters.

Among the usual security guarantees of hash functions we can ask for the halves H^i_s, for $i = 1, 2$, we require the hardness of computing many collisions on H^2_s. Bounding the probability of multi-collisions is reminiscent of deriving beyond-birthday security from symmetric (T)BC-based constructions of MAC or AEAD in the ideal-cipher model. In our construction, $\mathsf{H}^2_s(m) = v$ is used as a tweak and we want to have a probability that decreases fast in the number of additional colliding v-values. By controlling the probability of multi-collisions on H^2_s, we will have only few messages m_i in the adversarial view such that $\mathsf{H}_s(m_i) = u_i \| v$ for distinct u_i (otherwise we have a collision on the full H_s). That is, few possible preimages of F^v_k for any chosen v to compute a forgery.

We follow [7] to define a natural notion of multi-collision resistance before adapting it to the "half output" case as we require for our proofs.

Definition 8. *Let $\mathsf{H} : \mathcal{HK} \times \{0,1\}^* \to \{0,1\}^n$ be a hash function, and $\mu \geq 2$ be an integer. We say that H is μ-multi collision resistant if, for any efficient adversary A, there is a negligible function negl such that:*

$$\Pr_{\substack{s \leftarrow \mathcal{HK} \\ m_1, \ldots, m_\mu \leftarrow \mathsf{A}(s)}} \left[\forall i, j \in [\mu] : i \neq j \to m_i \neq m_j \wedge H_s(m_i) = H_s(m_j) \right] \leq \mathsf{negl}(n).$$

We further say that H is multi collision resistant if there is a value $\mu \geq 2$ such that H is μ-multi collision resistant.

If H is modeled as a random oracle, H is clearly μ-multi collision resistant for *any* $\mu \geq 2$. In the standard model, H might only be multi collision resistant from some lower bound, say $\mu \geq 10$ or $\mu \geq n/4$. Still, if H is μ-multi collision resistant it is also μ'-multi collision resistant, for any $\mu' \geq \mu$. We elaborate more on the possible behavior of μ in relation with a decreasing family of negligible functions to derive more concrete bounds in Section 4.4, after the security analysis.

We now simply define the *half multi-collision resistance* of $\mathsf{H}_s\colon \{0,1\}^* \mapsto \{0,1\}^{2n}$ as the multi-collision resistance of H_s^1 or H_s^2, where $\mathsf{H}_s = \mathsf{H}_s^1 \| \mathsf{H}_s^2$.

Definition 9. *Let* $\mathsf{H} : \mathcal{HK} \times \{0,1\}^* \to \{0,1\}^{2n}$ *be a hash function, and define* $\mathsf{H}^i : \mathcal{HK} \times \{0,1\}^* \to \{0,1\}^n$*, for* $i = 1, 2$*, such that for any* $s \leftarrow \mathcal{HK}$ *and any message* $m \in \{0,1\}^*$*, we have* $\mathsf{H}_s(m) = \mathsf{H}_s^1(m) \| \mathsf{H}_s^2(m)$*. We say that* H *is* half multi-collision resistant *for* $i = 1, 2$ *if* H^i *is multi-collision resistant. To avoid indexes, if* $i = 1$*, we also say that* H *is* upper*-half multi-collision resistant, and, if* $i = 2$*, that* H *is* lower*-half multi-collision resistant.*

Assuming that H satisfies this notion is a falsifiable assumption.

4.3 Security analysis of LR-MAC2

Unbounded leakage specification. Before assessing the sUF−L2 security of LR-MAC2, we have to define the leakage that an adversary can collect:

- $\mathsf{L}_M(m;k)$ returns $(y, \mathsf{LEval}(v,u;k_1), \mathsf{LEval}(v,y;k_2))$, where $u\|v \xleftarrow{n} \mathsf{H}_s(m)$ and the intermediate value $y = \mathsf{F}_{k_1}^v(u)$ is given.
- $\mathsf{L}_V(m,\tau;k)$ returns $(y, \tilde{y}, \mathsf{LEval}(v,u;k_1), \mathsf{LInv}(v,\tau;k_2))$, where $u\|v \xleftarrow{n} \mathsf{H}_s(m)$, and the intermediate values $y = \mathsf{F}_{k_1}^v(u)$ and $\tilde{y} = \mathsf{F}_{k_2}^{-1,v}(\tau)$ are given.

Our main result for LR-MAC2 is then formalized by the following theorem:

Theorem 2. *Let* $\mathsf{H} : \mathcal{HK} \times \{0,1\}^* \to \{0,1\}^{2n}$ *be a* $(t_1, \epsilon_{\mathsf{CR}(2n)})$*-collision resistant hash function and* $(t_1, \epsilon_{\mu\text{-}\mathsf{CR}(n)})$*-*$\mu$*-lower-half multi-collision resistant, for some* $\mu \geq 2$*. Let* F *be a* $(q_\mathsf{L}, q_\mathsf{Eval}, q_\mathsf{Inv}, t_2, \epsilon_{\mathsf{sUP\text{-}L2}})$*-strongly unpredictable with leakage* TBC*, then, LR-MAC2 is* $(q_\mathsf{L}, q_M, q_V, t, \epsilon)$*-*sUF−L2 *with:*

$$\epsilon \leq \epsilon_{\mathsf{CR}(2n)} + \epsilon_{\mu\text{-}\mathsf{CR}(n)} + 2\mu q_V \epsilon_{\mathsf{sUP\text{-}L2}},$$

where $q_\mathsf{LF} \leq q_\mathsf{LF}$*,* $q_\mathsf{Eval} \leq q_M + q_V + 1$*,* $q_\mathsf{Inv} \leq q_V$*,* $t_1 \leq t + (q_M + q_V + 1)t_\mathsf{H} + 2(q_M + q_V)(t_\mathsf{F} + t_{\mathsf{L}(\mathsf{F})})$*, and* $t_2 \leq t + (q_M + j + 1)t_\mathsf{H} + (q_M + q_V + 1)(t_\mathsf{F} + t_{\mathsf{L}(\mathsf{F})})$*.*

Below, we give a detailed sketch of the proof. We defer the full proof to Appendix C and we discuss the security bound in Section 4.4.

Proof (Sketch). To simplify our exposition, we assume the adversary gets the unbounded leakage without being explicit. Let (m,τ) be the first valid message-tag pair involved in a verification query which can be used by the adversary as a forgery. That is, the pair (m,τ) is *fresh* at the time of that verification query in the sense that it was never involved in an earlier verification query and (m,τ) is not the result of any previous tag-generation query for m. Given such a fresh and valid pair, let $u\|v \xleftarrow{n} \mathsf{H}_s(m)$, $y = \mathsf{F}_{k_1}^v(u)$ and $\tilde{y} = \mathsf{F}_{k_2}^{-1,v}(\tau)$. Below, we consider different cases depending on whether the TBC triples (v,u,y) for k_1 and $(v,\tilde{y},\tau)$ for k_2 have already been defined before the verification query of (m,τ) or not. We note that if the computation of $\tau = \mathsf{F}_{k_2}^v(\tilde{y})$ occured before (necessarily in a tag-generation query) our second triple is already defined.

F_1 The triples (v, u, y) for k_1 and $(v, \tilde{y}, \tau)$ for k_2 have not been defined yet.

F_2 The triple (v, u, y) for k_1 has already been defined, while $(v, \tilde{y}, \tau)$ for k_2 has not been defined yet.

F_3 Conversely, the triple (v, u, y) for k_1 has not been defined yet, while $(v, \tilde{y}, \tau)$ for k_2 has already been defined.

F_4 Both triples (v, u, y) for k_1 and $(v, \tilde{y}, \tau)$ for k_2 have already been defined. We split that case into three sub-cases:

- F_{41} Both triples (v, u, y) for k_1 and $(v, \tilde{y}, \tau)$ for k_2 have been defined in the same tag-generation or verification query.
- F_{42} The triple (v, u, y) for k_1 was defined in some query (of any type) precedent to the one in which $(v, \tilde{y}, \tau)$ was defined for k_2.
- F_{43} The triple $(v, \tilde{y}, \tau)$ for k_2 was defined in some query (of any type) precedent to the one in which (v, u, y) was defined for k_1.

We now give intuition about how we bound the probability of these events. The main difficulty arises in F_4, where would like to rely on the sUP–L2 security of the TBC. In F_{42} (resp., F_{43}), we would like to use the second-call triple $(v, \tilde{y}, \tau)$ (resp., the first-call triple (v, u, y)) as our winning prediction against the TBC. The trouble comes while we should avoid querying an evaluation/inversion that settles all the triples associated to (m, τ) as defined, in which case the attack will not be successful (to win the triple must not be defined). Fortunately, half-multi-collisions on v allow detecting the potential triples for which we have to be careful. Since their amounts are bounded by μ, unless H is not lower-half multi-collision resistant, we only get an additional factor μ in $\epsilon_{\mathsf{sUP\text{-}L2}}$ for each case.

F_1 We build a sUP–L2 adversary against the TBC, conceptually against the "second call" F_{k_2}. It picks $k_1 \xleftarrow{\$} \mathcal{K}$ for itself and simulates the computation of F_{k_2} by querying its own TBC oracle. In the verification query (m, τ), it computes y itself and outputs (v, y, τ) as its prediction. Given that (m, τ), the probability that this event occurs is bounded by $\epsilon_{\mathsf{sUP\text{-}L2}}$.

F_2 We build a sUP–L2 adversary against the TBC as in the case of F_1.

F_3 We build a sUP–L2 adversary against the TBC, conceptually against the "first call" F_{k_1}. It picks $k_2 \xleftarrow{\$} \mathcal{K}$ for itself and simulates the computation of F_{k_1} by querying its own TBC oracle. In the verification query (m, τ), it computes $\tilde{y}$ itself and outputs $(v, u, \tilde{y})$ as its prediction. Given that (m, τ), the probability that this event occurs is bounded by $\epsilon_{\mathsf{sUP\text{-}L2}}$.

F_4 This case is less straightforward. We deal with its sub-cases independently.

- F_{41} Let (m', τ') be the message-tag pair involved in the earlier query that defines (v, u, y) and $(v, \tilde{y}, \tau)$. Then, $\tau' = \tau$ (regardless of the query type).
  - F_{411} If $m^i = m$, it contradicts the freshness of (m, τ). This case is void.
  - F_{412} If $m^i \neq m$, since the defined triples implies $u \| v \xleftarrow{n} \mathsf{H}(m')$, we found a collision. The probability that this event occurs is bounded by ϵ_{CR}.
- F_{42} Let (m^i, τ^i) be the message-tag pair involved in the earlier query that defines (v, u, y) for k_1 and (m^j, τ^j) be the message-tag pair involved in a subsequent query that defines $(v, \tilde{y}, \tau)$ for k_2, and both (regardless of their respective type of query) occurring before our (m, τ). By the

meaning of the triples, we have $u\|v \xleftarrow{n} \mathsf{H}(m^i)$ and $\tau^j = \tau$, and we can assume that $m^i = m$ as we already dealt with collisions. However, we note that $\tau^i \neq \tau$ and $m^j \neq m$ as otherwise it would contradict the freshness of (m, τ). So (m^i, τ^i) must be an invalid pair in a verification query as well as (m^j, τ^j) (if it is valid it can only be a tag-generation query by definition of (m, τ), but then $u^j = \mathsf{F}_{k_1}^{v,-1} \circ \mathsf{F}_{k_2}^{v,-1}(\tau^j) = \mathsf{F}_{k_1}^{v,-1} \circ \mathsf{F}_{k_2}^{v,-1}(\tau) = u$).[11] This implies that we already have a lower-half 2-multi-collision on v. Indeed, we see that $\mathsf{H}_s^2(m^i) = \mathsf{H}_s^2(m^j)$ with $m^i \neq m^j$.

Then, to avoid defining the second-triple $(v, \tilde{y}, \tau)$ we must use it as a winning prediction against the TBC at the time the query (m^j, τ^j) is made. Moreover, the situation will be the same with any subsequent query (m^j, τ^j) before (m, τ) where the second-triple is given by $(v, \tilde{y}, \tau)$, i.e., $v^l = v$, $\tau^l = \tau$ and distinct m^l (otherwise it is a repeating query).

We can bound the probability of all these events given (m^i, τ^i). Each time a new verification query is made for some (m^l, τ^l) such that $m^i \neq m^l$ and $v^i = v^l$ (we include $l = j$ here), for the at most $\mu - 1$ possible cases, we make an hybrid on the ordered such l, and a reduction to the $\mathsf{sUP\text{–}L2}$ experiment against the TBC in the "second call" by picking k_1 and outputting the prediction (v^i, y^i, τ^l) (while the TBC oracle is called for all the previous verification queries to emulate the second MAC call).

F_{43} We are in the same case as case F_{42} with the difference that the query (m^j, τ^j) defining $(v, \tilde{y}, \tau)$ for k_2 is made before the query (m^i, τ^i) defining (v, u, y) for k_1. Since our argument showing that these queries in F_{42} are of verification type and the input pairs are invalid does not depend on which query happens before, we are exactly in the dual situation where we somehow switched the first-call and the second-call.

We can bound the probability of all these events given (m^j, τ^j). Each time a new verification query is made for some (m^l, τ^l) such that $m^l \neq m^j$ and $v^l = v^j$ (we include $l = i$ here), for the at most $\mu - 1$ possible cases, we make an hybrid on the ordered such l, and a reduction to the $\mathsf{sUP\text{–}L2}$ experiment against the TBC in the "first call" by picking k_2 and outputting the prediction $(v^j, u^l, \tilde{y}^j)$ (while the TBC oracle is called for all the previous verification queries to emulate the second MAC call).

In the full proof, we first deal with the collision resistance of H_s and the μ-multi-collision resistance of H_s^2. For the events F_{42} and F_{43} we of course do not know when we will be in face of the queries (m^i, τ^i) and (m^j, τ^j) which could define the triples of the the first potential forgery (m, τ). However, by considering all the verification queries as potentially being (m^i, τ^i) or (m^j, τ^j) that defines the first triple of (m, τ), we cover all the cases with a term like $2\mu q_V \epsilon_{\mathsf{sUP\text{–}L2}}$. That way, we also cover the case that (m, τ) might be any of the verification queries. Indeed, once we analyze a potential (m^i, τ^i) or (m^j, τ^j), we will consider all the

[11] The choice to reuse v as the only tweak of the TBC in our design is crucial. With distinct tweaks the security bound will be much more loose. Among other advantages, here, we only have to deal with the verification queries.

next verification queries sharing v^i or v^j among which all the possible (m, τ) lie when F_{42} or F_{43} occur. In the remaining case, it is straightforward to deal with F_1, F_2 and F_3. Overall, we find $[2(\mu-1)q_V + q_V + 1]\epsilon_{\mathsf{sUP\text{-}L2}}$ for the TBC term.

4.4 Deriving concrete bounds

Deriving concrete bounds from asymptotic security is not an easy task in the standard model. However, we can argue why it is realistic to assume the existence of building blocks which will confer high security to LR-MAC2. Since the sUP–L2 of the TBC can be argued in the same lines as in the rest of the paper, we next focus on the stronger properties that we require for the hash function.

Multi-collision resistant hashing. The security bound shows that it is reasonable to take $\mu \in O(n)$ without blowing up the term related to the unpredictability of the TBC. Moreover, increasing μ can only decrease the probability of finding more and more multi-collisions. Indeed, μ-multi collision resistance implies $(\mu+1)$-multi collision resistance. And, even it the definition does not imply that the negligible probability decreases when μ increases, it is reasonable to assume so for "good" hash functions, as for SHA2 or SHA3.

In the ROM, it is a well-known fact that n-multi collision resistance allows up to $q \approx 2^n/n$ hash queries. Of course, we will not assume in the standard model that H remains secure up to that number of evaluations. However, it might be the case that H is $3n$-multi collision resistant up to $q \approx 2^{96}$, which is much less demanding and fully compatible with our other terms in our security bound.

As another justification, Berti et al. [9, Lemma 5] proved that the Merkle-Damgård iteration of the Hirose's DBL compression function [26] gives rise to a half-multi-collision resistant hash function in the ideal cipher model.

Multi-collision independence. By more closely mimicking the behavior of random functions, we derive another relaxed notion that could be realized without idealized assumption on half outputs of H. This is a stronger definition than the (half-) multi-collision resistance notion, but it offers a way to more accurately model a decreasing negligible functions' family depending on the multi-collision parameter μ. This allows us to soundly analyze how the negligible functions related to a (half-) μ-multi-collision resistant hash function H might become better and better as μ grows, as expected from any "good" hash function, and without relying on any ideal argument. Of course, the decreasing rate in μ can be much slower than a truly random function but our security bounds of LR-MAC2 shows how comfortable we are to increase μ even in a leakage setting.

First, we recall a result of [35] showing that for a random function with 2^n possible outputs, the multi-collision event $\mathsf{MultiColl}(2^n, q) \geq \mu$ that at least $\mu \geq 2$ inputs give a same output among a total of q function evaluations satisfies:

$$\Pr[\mathsf{MultiColl}(2^n, q) \geq \mu] \leq \frac{1}{2^{n(\mu-1)}} \binom{q}{\mu}.$$

Since the right-hand side is also bounded by $q^\mu/2^{n(\mu-1)}$, we can simplify this result by saying that any additional collision comes at a marginal probability of $\approx q/2^n$ starting from the standard collision bound $q^2/2^n$, where $\mu = 2$.

Definition 10. *Let* $\mathsf{H} : \mathcal{HK} \times \{0,1\}^* \to \{0,1\}^n$ *be a hash function. We say that* H *is* θ*-powerwise collision independent if, for* $s \leftarrow \mathcal{HK}$*, the probability that an efficient adversary* $\mathsf{A}(s)$ *making at most* q *hash evaluations computes at least* μ *distinct* $m_1, \ldots, m_\mu \in \{0,1\}^*$ *such that* $\mathsf{H}_s(m_1) = \ldots = \mathsf{H}_s(m_\mu)$ *is bounded by:*

$$2^n \left(\frac{q^{1+\theta}}{2^n}\right)^\mu,$$

with $\mu \geq 2$*,* $0 \leq \theta \leq 1$*. The factor* $q^{1+\theta}/2^n$ *is the marginal collision probability and* θ *is called the non-idealization power. We say that* H *is* multi-collision independent *if it is* θ*-powerwise collision independent for some* $\theta \in (0,1)$*.*

If the non-idealization power is null, the hash function is a random function and the marginal collision probability matches the one discussed above.

We stress that finding a μ-multi-collision for any $\mu \geq 2$ gives the same chance of finding "one more" collision in the sense of a $(\mu+1)$-multi-collision. Said otherwise, the marginal collision probability is independent of the number of multi-collisions already found, hence the name. For instance, finding more multi-collisions does not help in finding even more multi-collisions. This might be seen as a strong collision resistant flavor as the hash function should resist further collision attacks even if some previous attacks already succeeded. However, standard hash functions should fulfill such a collision resilient condition. At least, they have to offer a graceful degradation and, in particular, given a collision it should not become easy to find some others. Still, we require here that the degradation factor be constant, given the security parameter n and the non-idealization power θ. But we might only have a middle non-idealization power $\theta = 1/2$. In that case, the collision probability (i.e., the 2-multi-collision) is only upper-bounded by $q^3/2^n$ ensuring collision security up to $q \approx 2^{42}$, for $n = 128$. Nevertheless, we can reach the 20-multi-collision security up to $q \approx 2^{81}$. By letting $\theta = 1/\rho$, we can reach the n-multi-collision security up to $q \approx 2^{n\rho/(1+\rho)}$.

We now simply define *half-multi-collision independence* of $\mathsf{H}_s \colon \{0,1\}^* \mapsto \{0,1\}^{2n}$ as the multi-collision independence of H_s^1 or H_s^2, where $\mathsf{H}_s = \mathsf{H}_s^1 \| \mathsf{H}_s^2$.

Definition 11. *Let* $\mathsf{H} : \mathcal{HK} \times \{0,1\}^* \to \{0,1\}^{2n}$ *be a hash function, and define* $\mathsf{H}^i : \mathcal{HK} \times \{0,1\}^* \to \{0,1\}^n$*, for* $i = 1, 2$*, such that for any* $s \leftarrow \mathcal{HK}$ *and any message* $m \in \{0,1\}^*$*, we have* $\mathsf{H}_s(m) = \mathsf{H}_s^1(m) \| \mathsf{H}_s^2(m)$*. We say that* H *is* half-multi-collision independent *for* $i = 1, 2$ *if* H^i *is* multi-collision independent*. To avoid indexes, if* $i = 1$*, we also say that* H *is* upper-*half-multi-collision independent, and, if* $i = 2$*, that* H *is* lower-*half-multi-collision independent.*

Armed with this stronger theoretical background, we can re-evaluate the security of LR-MAC2 in the light of the non-idealization power θ, assuming that H is lower-half-multi-collision independent (where H_s^2 outputs tweaks). Let us

first recall the security bound $\epsilon_{\mathsf{CR}(2n)} + \epsilon_{\mu\text{-}\mathsf{CR}(n)} + 2\mu q_V \epsilon_{\mathsf{sUP\text{-}L2}}$ of Theorem 2, and assume that $\epsilon_{\mathsf{sUP\text{-}L2}}$ supports 2^{85} (online) queries, for $n = 128 = 2^7$. Setting $q_V = 2^{75}$ then allows us to take $\mu = 2^8$. That means that if the number of evaluation of H remains below 2^{90}, the non-idealization power could be $\theta = 2/5$ for H_s^2 since $\epsilon_\mu \leq 2^{128}(2^{90\cdot 7/5}/2^{128})^{256} < 2^{-128}$, even if H_s^2 would only be (2-multi-) collision resistant up to $2^{63/(1+\theta)} = 2^{45}$ hash evaluations.

So overall, we conclude this section by observing that LR-MAC2 could provide strong leakage resilience guarantees. Yet, on the one hand, it requires stronger assumptions which, despite not idealized, are less standard than the collision resistance that LR-MAC1 requires. On the other hand, it also requires two calls to a TBC with n-bit tweaks which, as mentioned in introduction, should in general be more expensive than one call to a TBC with $2n$-bit tweaks if specialized constructions can be used. We note there are constructions that achieve the same leakage-resilience guarantees without half collision resistance, but the best solution we are aware of (given in Appendix B, Figure 5 or completeness) requires three TBC calls for this purpose. So despite theoretically interesting, we believe these results also amplify the pragmatic interest of using large tweaks.

5 Analysis of HTBC

In this section, we finally show that the popular Hash-then-PRF MAC construction is provable without idealized assumptions in a leakage setting. For consistency with the LR-MAC1 and LR-MAC2 constructions, we focus on the Hash-then-TBC scheme described in Figure 3 and specified in Algorithm 3.

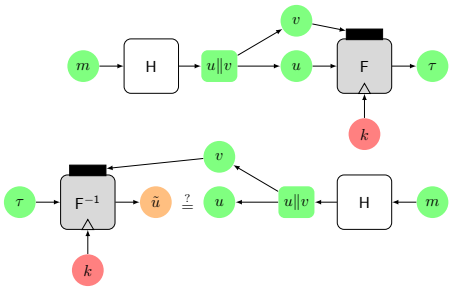

Fig. 3: Hash-then-TBC.

5.1 A first analysis in the standard model

The security proof requires the hash H to be collision resistant and range-oriented preimage resistant. A stronger (and less standard) assumption is *weak image set computable*, that is, there exists a PPT algorithm M that outputs the set

Algorithm 3 The HTBC algorithm.

It uses a strongly protected TBC $\mathsf{F} : \mathcal{K} \times \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ and a hash function $\mathsf{H} : \mathcal{HK} \times \{0,1\}^n \to \{0,1\}^{2n}$.

- Gen
  - $k \xleftarrow{\$} \mathcal{K}$
  - $s \xleftarrow{\$} \mathcal{HK}$
- $\mathsf{Mac}_k(m)$:
  - $u \| v = \mathsf{H}_s(m)$
  - $\tau = \mathsf{F}_k^v(u)$
  - Return τ
- $\mathsf{Vrfy}_k(m, \tau)$:
  - $u \| v = \mathsf{H}_s(m)$
  - $\tilde{u} = \mathsf{F}_k^{v,-1}(\tau)$
  - If $\tilde{u} == u$ Return $\top$
  - Else Return $\bot$

$\mathcal{WP}(s, \mathsf{A})$ defined in Eq. (1) for any s and A. By Lemma 1, for range-oriented preimage resistant hash functions it is feasible to output the set $\mathcal{WP}(s, \mathsf{A})$ in PPT. We will serve more intuitions on this assumption at the end of this section.

The assumption of *weak image set computability* admittedly renders the concrete security analysis hard to interpret. As a result, below we eschew the concrete security approach (which is followed by Theorems 1 and 2) in favor of the *asymptotic approach*. We justify this choice by the fact that the goal of this section is to show a theoretical possibility, potentially opening a path towards more advanced and concrete analyzes in the future.

Theorem 3. *Let the hash function* H *be collision resistant, range-oriented preimage resistant, and weak image set computable. Let* F *be a* TBC *that is* $\mathsf{sUP\text{-}L2}$*. Then,* HTBC *is* $\mathsf{sUF\text{-}L2}$*-secure in the unbounded leakage model.*

Proof. In the unbounded leakage model, all the intermediate computations leak (except the long-term key). Wlog, we assume that the forgery adversary A has been "normalized". That is, before making every oracle query, A outputs the list of all images $\mathcal{S}(s) \subseteq \{0,1\}^n$ for which she knows the preimage. Clearly, this cannot decreases the success probability of A. Moreover, this only induces a polynomial blow-up in A's running time: $\mathcal{S}(s)$ must be polynomial, as otherwise A gets a superpolynomial number of input/output relations $\mathsf{H}_s(m) = y$ and would be able to break the collision resistance of H.

To prove the theorem, we follow a sequence of games.

Game 0. Let Game 0 be the $\mathsf{sUF\text{-}L2}$ game where the adversary A^{L} tries to produce a forgery when she plays against HTBC. Let E_0 be the event that the adversary wins the game; that is, the output of the game is 1.

Game 1. Game 1 is Game 0, except that we abort if there is a full collision in the hash function. Clearly, Game 0 and Game 1 are identical if the following event

HC (*Hash collision*) does not happen: $HC := \{\exists i, j \in \{1, ..., q_M\} \cup \{1, ..., q_V + 1\}$ with $i \overset{\%}{\neq} j$ s.t. $h^i = h^j$ and $m^i \neq m^j\}$ (see Footnote 9). Since H is collision resistant, we have $\Pr[\text{Game } 0 \Rightarrow 1] \approx \Pr[\text{Game } 1 \Rightarrow 1]$.

Linking to unpredictability. To complete the argument, we exhibit a predictor B against F_k using an adversary A that forges in Game 1. In detail, B picks a hash seed s uniformly at random, passes s to A and simulates Game 1 against A. B also picks an index $\ell \xleftarrow{\$} \{1, ..., q_v + 1\}$ at random. Assume that A outputs the forgery (m^*, τ^*) at the end of the game, where $\mathsf{H}_s(m^*) = u^* \| v^*$.

Now, if Game 1 outputs 1, i.e., $\mathsf{VrfyL}_k(m^*, \tau^*)$ returns $\top$ at the end, then the probability that the ℓ-th verification query $\mathsf{VrfyL}_k(m^{(\ell)}, \tau^{(\ell)})$ constitutes the first time $(\mathsf{F}_k^{v^*})^{-1}(\tau^*)$ is queried is at least $1/q_v$. It can be seen that either of the following bad events necessarily occurred during the game:

- Case 1: intuitively, at some time, the oracle $(\mathsf{F}_k^{v^*})^{-1}(\tau^*)$ returns a target u^* and A later solves its preimage $\mathsf{H}_s(m^*) = u^* \| v^*$; or,
- Case 2: intuitively, at some time, A has been aware of the relation $\mathsf{H}_s(m^*) = u^* \| v^*$, and the oracle $(\mathsf{F}_k^{v^*})^{-1}(\tau^*)$ later returns a target $u' = u^*$.

But B cannot predict which case will be encountered. Fortunately, in either case the number of candidates for $(\mathsf{F}_k^{v^*})^{-1}(\tau^*)$ is polynomial, and thus B can simply take a union. In more detail:

If the first case is encountered, then the concatenation $u^* \| v^*$ of the returned target $u^* = (\mathsf{F}_k^{v^*})^{-1}(\tau^*)$ and the tweak v^* necessarily falls in the set $\mathcal{WP}(s, \mathsf{A})$ (otherwise A won't be able to solve the preimage). Then, since we assumed that the set $\mathcal{WP}(s, \mathsf{A})$ is computable given s and A, the predictor B could make a guess u' among $\mathcal{WP}(s, \mathsf{A})$ and take (v^*, u', τ^*) as a forgery for F_k.

If the second case is encountered, then as we assumed that A always outputs the list $\mathcal{S}(s)$ of all images that she knows the preimage, the predictor B could make a guess u' among $\mathcal{S}(s)$ and takes (v^*, u', τ^*) as a forgery for F_k.

By the above, the predictor B first runs the algorithm M to obtain $\mathcal{WP}(s, \mathsf{A})$, and then reads the set $\mathcal{S}(s)$ from A's outputs. B then picks $u' \xleftarrow{\$} \mathcal{WP}(s, \mathsf{A}) \cup \mathcal{S}(s)$ uniformly and outputs (v^*, y', τ^*) as a forgery for F_k.

Summing over the two cases, the success probability of B is at least:

$$\begin{aligned}
& \Big(\frac{\Pr[\text{Game } 1 \Rightarrow 1]}{q_v + 1}\Big) \cdot \Big(\frac{\Pr[\text{Case } 1]}{|\mathcal{WP}(s, \mathsf{A})| + |\mathcal{S}(s)|} + \frac{\Pr[\text{Case } 2]}{|\mathcal{WP}(s, \mathsf{A})| + |\mathcal{S}(s)|}\Big), \\
\geq\ & \Big(\frac{\Pr[\text{Game } 1 \Rightarrow 1]}{q_v + 1}\Big) \cdot \Big(\Pr[\text{Case } 1] + \Pr[\text{Case } 2]\Big) \cdot \frac{1}{|\mathcal{WP}(s, \mathsf{A})| + |\mathcal{S}(s)|}, \\
=\ & \frac{\Pr[\text{Game } 1 \Rightarrow 1]}{(q_v + 1)(|\mathcal{WP}(s, \mathsf{A})| + |\mathcal{S}(s)|)}. \qquad (3)
\end{aligned}$$

By the unpredictability assumption on F, the final term $\Pr[\text{Game } 1 \Rightarrow 1]/(q_v + 1)(|\mathcal{WP}(s, \mathsf{A})| + |\mathcal{S}(s)|)$ should be negligible, which means $\Pr[\text{Game } 1 \Rightarrow 1] = \mathsf{negl}(n)$. This complete the proof. □

Discussion. To facilitate understanding, we provide further intuition on the proof of Theorem 3. A first impression may suggest the infeasibility to decide and output the weak set $\mathcal{WP}(s, \mathsf{A})$ since the seed space is exponential, which (seems) to gain support from the formalism of collision resistant hashing. However, this intuition is incorrect: the seed space being exponential does not *necessarily* mean the computation is infeasible. In particular, it may be easy to derive $\mathcal{WP}(s, \mathsf{A})$ from the seed, or even seed-independent (i.e., the algorithm does not necessarily need to keep a table for all the seeds) as we now detail.

First, assume using a keyed random oracle $\mathsf{RO} : \mathcal{HK} \times \{0,1\}^* \to \{0,1\}^{2n}$ as the hash function. In this case, $\Pr_{\mathsf{A}(s,y)\Rightarrow m}\left[\mathsf{RO}_s(m) = y\right]$ is negligible for all $y \in \{0,1\}^{2n}$, meaning that $\mathcal{WP}(s, \mathsf{A}) = \emptyset$. By this, the probability that Case 1 in the proof occurs is negligible (which is indeed the case, as the probability is in $O(q/2^n)$, with q the number of adversarial RO queries). On the other hand, the "normalized" adversary A simply outputs its RO query/response list before the ℓ-th verification query, and this could be used to predict $(\mathsf{F}_k^v)^{-1}(\tau)$.[12].

Let us now assume using a keyed SHA3 variant (denoted sha) producing 256-bit hash digests. According to known cryptanalytic results, for any A with running time t, $\Pr_{\mathsf{A}(s,y)\Rightarrow m}\left[\mathsf{sha}_s(m) = y\right] = O(t/2^{256})$ for all $y \in \{0,1\}^{256}$, meaning that $\mathcal{WP}(s, \mathsf{A})$ remains $\emptyset$ in some sense.

On the one hand, it seems that the number of SHA3 input/output pairs that could be derived in time t is only $O(t)$, and the "normalized" adversary A simply outputs a list of $O(t)$ input/output pairs. On the other hand, the double-block-length hash function Tandem-DM has a "weak" image 0 that is easily invertible [3]. If Tandem-DM is used, then $\mathcal{WP}(s, \mathsf{A}) = \{0\}$, the size of which is just 1. Further consider the keyed hash $\text{KeyedTDM}_s(M) = \text{Tandem-DM}(M) \oplus s$. Despite being contrived, for this instructive example the weak set $\mathcal{WP}(s, \mathsf{A}) = \{s\}$ which is seed-dependent but very easy to derive.

Note that we do not need the hash function to be non-malleable. The proof follows even if the hash admits the weakness that $\mathsf{H}_s(m \oplus \delta_1) = \mathsf{H}_s(m \oplus \delta_1) \oplus \delta_2$ for *a specific pair* of differences (δ_1, δ_2) and *any* m: in this case, the number of "known" input/output pairs remains in $O(t)$. We additionally remark that:

- the weak image set computability assumption doesn't contradict the range-oriented preimage resistant assumption, because the latter concerns with random images that are unlikely to fall in $\mathcal{WP}(s, \mathsf{A})$;
- this assumption doesn't contradict the collision resistant assumption either, because weak images don't necessarily have multiple preimages.

Finally, we note that our proof approach easily extends to the constructions of Hash-then-block cipher of [11] and Hash-then-PRF (enhanced with the leakage-resilient value comparison tricks [**?**]). It is also intriguing to ask whether more "common" assumptions on hashing could suffice, e.g., even the UCE hashing [5] or some weak form of correlation intractable hashing [13,12]. We leave the investigation of these alternative approaches for future investigations.

[12] The latter trick was also used in [8].

5.2 Towards another analysis

As an opening, we eventually discuss whether adaptating the security proofs of [8] could lead to another analysis. The simplest solution for this purpose is to find the appropriate assumption on H that could hold in the standard model to replace the random oracle. This is also what is done in the previous analysis but we next suggest another (stronger) possibility that is simple to express.

Essentially, what we need from the hash function H is that any *targeted* image y that has never been defined yet by computing $\mathsf{H}(x)$ for some x is still unattainable. Of course, by computing $\mathsf{H}(x)$ for a fresh input x we always attain a new image with overwhelming probability, unless we find a collision. Obviously, all the undefined images cannot be our targets. So, the main difficulty is to define a natural meaning of *targets* that can be chosen *adaptively* by the adversary. This is where the new definition will anyway be less standard than usual pre-image resistance. But this is precisely what we need in the security proof of HTBC to adapt [8]. Indeed, let $\tilde{u} = \mathsf{F}_k^{v,-1}(\tau)$, and let us consider again the cases where either $\tilde{u}\|v$ was already computed from $\mathsf{H}(m)$ for some m or it was not. In the first case, if we know that $\mathsf{H}(m) = \tilde{u}\|v$ has already been computed in the unforgeability game, we can easily build an adversary against TBC and compute a prediction. In the second case, if we know that $\tilde{u}\|v$ was never defined as an output of H, then we want to define $\tilde{u}\|v$ as a target. That means that if we know which hashing has already been computed, we can adaptively define a set of targets of polynomial size. Since the number of TBC calls is bounded by the number of queries against the unforgeability of HTBC, knowing which value is already a defined image of H is precisely what we need, and what is done in the random oracle model. Still, we do not need to have uniformly random output.

As a consequence, as long as H is half-multi-collision independent or even half-multi-collision resistant as defined in Section 4, since the proof in [8] already needs such kind of assumption even if it comes for free from the random oracle, the proof can go through if we require H to satisfy the notion of *essentially pre-image resistance* which precisely models the hardness of computing a pre-image of a single target among any polynomial set of targets that can be adaptively updated by the adversary before trying to find it.

Essentially-pre-image resistance would simply model which hashing is already defined or not and maintain the list of targets that the adversary announced she will try to attack. To do so, even if the hash function is a public-key primitive, we should demand that computing hash values requires "querying" H even if the hash values are computed *faithfully* (without any other restriction about the distribution of the outputs). Hashing becomes interactive in that model but *only* for the sake of defining fresh targets and nothing else. The interaction does not occur in practice: it is just a way to soundly define what are the fresh targets and to avoid trivial attacks in the standard model.

It is easy to see that for any Turing Machine A without interaction with H, we can build a Turing Machine A' that explicitly returns all the inputs it uses for H, and directly after the output of that computation. Based on this discussion, we can see that the security of HTBC can be essentially good.

6 Conclusions

This paper describes and analyzes three leakage-resilient MACs that can be proven based on minimum physical assumptions (namely the unpredictability of a TBC with leakage) and more or less standard (anyway non idealized) black box assumptions for their hash part. We believe these results make an important step in building cryptographic primitives enabling strong security against side-channel attacks at limited cost, by leveraging the leveled implementation concept. The constructions we analyze range from pragmatic to theoretically more involved. The difficulty to maintain tight bounds with minimum assumptions when getting rid of the design twists used in our pragmatic solution (i.e., large tweaks for the TBC and a verification process that checks the inverse TBC against a constant) confirms that selecting good assumptions is critical for such constructions, as generally observed in cryptography but possibly even more with leakage. Avoiding idealized assumptions is important in this respect, since they make it impossible to translate security proofs and bounds into concrete requirements for the engineers implementing the corresponding schemes.

These results suggest different tracks for further investigations. In view of the limited overheads that its design twists imply, further instantiating the LR-MAC1 construction and designing a TBC with large tweaks that is suited to side-channel countermeasures appears as a practically relevant goal. In parallel, investigating whether the performances' and assumptions' gaps between LR-MAC1 and the LR-MAC2 or HTBC constructions can be tightened is an interesting question as well (for example using some of the directions outlined at the end of Section 5). Eventually, extending our results to sponge-based constructions (e.g., ISAP-MAC [15]) is another challenge that would deserve attention: we expect that such an analysis will require some adaptation and different assumptions. For example, secure authentication in this case cannot directly work in the unbounded leakage model, as per the recent work of Dobraunig and Menning [17]. The same holds for extending the quest for leakage-resilient constructions without idealized assumptions from MACs to (authenticated) encryption schemes.

Acknowledgments

We thank the ASIACRYPT 2021 reviewers for their insightful feedback. Francesco Berti was founded by the Emmy Noether Program FA 1320/1-1 of the German Research Foundation (DFG). Chun Guo was supported in parts by the Program of Taishan Young Scholars of the Shandong Province, the Program of Qilu Young Scholars (Grant No. 6158008996 3177) of Shandong University, the National Natural Science Foundation of China (Grant No. 62002202), and the Shandong Nature Science Foundation of China (Grant No. ZR2020MF053). Thomas Peters and François-Xavier Standaert are research associate and senior research associate of the Belgian Fund for Scientific Research (F.R.S.-FNRS). This work has been funded in parts by the EU through the ERC project SWORD (724725).

References

1. J. H. An and M. Bellare. Constructing vil-macsfrom fil-macs: Message authentication under weakened assumptions. In *CRYPTO*, volume 1666 of *LNCS*, pages 252–269. Springer, 1999.
2. E. Andreeva and M. Stam. The symbiosis between collision and preimage resistance. In L. Chen, editor, *Cryptography and Coding - 13th IMA International Conference, IMACC 2011, Oxford, UK, December 12-15, 2011. Proceedings*, volume 7089 of *LNCS*, pages 152–171. Springer, 2011.
3. F. Armknecht, E. Fleischmann, M. Krause, J. Lee, M. Stam, and J. P. Steinberger. The preimage security of double-block-length compression functions. In D. H. Lee and X. Wang, editors, *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *LNCS*, pages 233–251. Springer, 2011.
4. M. Azouaoui, D. Bellizia, I. Buhan, N. Debande, S. Duval, C. Giraud, É. Jaulmes, F. Koeune, E. Oswald, F. Standaert, and C. Whitnall. A systematic appraisal of side channel evaluation strategies. In *SSR*, volume 12529 of *LNCS*, pages 46–66. Springer, 2020.
5. M. Bellare, V. T. Hoang, and S. Keelveedhi. Instantiating random oracles via uces. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, volume 8043 of *LNCS*, pages 398–415. Springer, 2013.
6. D. Bellizia, O. Bronchain, G. Cassiers, V. Grosso, C. Guo, C. Momin, O. Pereira, T. Peters, and F. Standaert. Mode-level vs. implementation-level physical security in symmetric cryptography - A practical guide through the leakage-resistance jungle. In *CRYPTO (1)*, volume 12170 of *LNCS*, pages 369–400. Springer, 2020.
7. I. Berman, A. Degwekar, R. D. Rothblum, and P. N. Vasudevan. Multi-collision resistant hash functions and their applications. In J. B. Nielsen and V. Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *LNCS*, pages 133–161. Springer, 2018.
8. F. Berti, C. Guo, O. Pereira, T. Peters, and F. Standaert. Strong authenticity with leakage under weak and falsifiable physical assumptions. In *Inscrypt*, volume 12020 of *LNCS*, pages 517–532. Springer, 2019.
9. F. Berti, C. Guo, O. Pereira, T. Peters, and F. Standaert. Tedt, a leakage-resist AEAD mode for high physical security applications. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):256–320, 2020.
10. F. Berti, F. Koeune, O. Pereira, T. Peters, and F. Standaert. Ciphertext integrity with misuse and leakage: Definition and efficient constructions with symmetric primitives. In *AsiaCCS*, pages 37–50. ACM, 2018.
11. F. Berti, O. Pereira, T. Peters, and F. Standaert. On leakage-resilient authenticated encryption with decryption leakages. *IACR Trans. Symmetric Cryptol.*, 2017(3):271–293, 2017.
12. R. Canetti, Y. Chen, L. Reyzin, and R. D. Rothblum. Fiat-shamir and correlation intractability from strong kdm-secure encryption. In J. B. Nielsen and V. Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I*, volume 10820 of *LNCS*, pages 91–122. Springer, 2018.

13. R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.
14. J. P. Degabriele, C. Janson, and P. Struck. Sponges resist leakage: The case of authenticated encryption. In *ASIACRYPT (2)*, volume 11922 of *LNCS*, pages 209–240. Springer, 2019.
15. C. Dobraunig, M. Eichlseder, S. Mangard, F. Mendel, B. Mennink, R. Primas, and T. Unterluggauer. Isap v2.0. *IACR Trans. Symmetric Cryptol.*, 2020(S1):390–416, 2020.
16. C. Dobraunig and B. Mennink. Leakage resilience of the duplex construction. In *ASIACRYPT (3)*, volume 11923 of *LNCS*, pages 225–255. Springer, 2019.
17. C. Dobraunig and B. Mennink. Leakage resilient value comparison with application to message authentication. In *EUROCRYPT (2)*, volume 12697 of *LNCS*, pages 377–407. Springer, 2021.
18. Y. Dodis and J. P. Steinberger. Message authentication codes from unpredictable block ciphers. In *CRYPTO*, volume 5677 of *LNCS*, pages 267–285. Springer, 2009.
19. Y. Dodis and J. P. Steinberger. Domain extension for macs beyond the birthday barrier. In *EUROCRYPT*, volume 6632 of *LNCS*, pages 323–342. Springer, 2011.
20. S. Dziembowski and K. Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302. IEEE Computer Society, 2008.
21. B. Fuller and A. Hamlin. Unifying leakage classes: Simulatable leakage and pseudoentropy. In *ICITS*, volume 9063 of *LNCS*, pages 69–86. Springer, 2015.
22. D. Goudarzi and M. Rivain. How fast can higher-order masking be in software? In *EUROCRYPT (1)*, volume 10210 of *LNCS*, pages 567–597, 2017.
23. C. Guo, O. Pereira, T. Peters, and F. Standaert. Towards low-energy leakage-resistant authenticated encryption from the duplex sponge construction. *IACR Trans. Symmetric Cryptol.*, 2020(1):6–42, 2020.
24. C. Guo, F. Standaert, W. Wang, and Y. Yu. Efficient side-channel secure message authentication with better bounds. *IACR Trans. Symmetric Cryptol.*, 2019(4):23–53, 2019.
25. C. Hazay, A. López-Alt, H. Wee, and D. Wichs. Leakage-resilient cryptography from minimal assumptions. In *EUROCRYPT*, volume 7881 of *LNCS*, pages 160–176. Springer, 2013.
26. S. Hirose. Some plausible constructions of double-block-length hash functions. In M. J. B. Robshaw, editor, *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, volume 4047 of *LNCS*, pages 210–225. Springer, 2006.
27. J. Jean, I. Nikolic, and T. Peyrin. Tweaks and keys for block ciphers: The TWEAKEY framework. In *ASIACRYPT (2)*, volume 8874 of *LNCS*, pages 274–288. Springer, 2014.
28. J. Jean, I. Nikolic, T. Peyrin, and Y. Seurin. Deoxys v1. 41. *CAESAR Competition, Final Portfolio*, 2016.
29. J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.
30. J. Krämer and P. Struck. Leakage-resilient authenticated encryption from leakage-resilient pseudorandom functions. In *COSADE*, volume 12244 of *LNCS*, pages 315–337. Springer, 2020.
31. D. P. Martin, E. Oswald, M. Stam, and M. Wójcik. A leakage resilient MAC. In *IMACC*, volume 9496 of *LNCS*, pages 295–310. Springer, 2015.
32. S. Micali and L. Reyzin. Physically observable cryptography (extended abstract). In *TCC*, volume 2951 of *LNCS*, pages 278–296. Springer, 2004.

33. O. Pereira, F. Standaert, and S. Vivek. Leakage-resilient authentication and encryption from symmetric cryptographic primitives. In *CCS*, pages 96–108. ACM, 2015.
34. J. Schipper. Leakage-resilient authentication. Master's thesis, 2011.
35. K. Suzuki, D. Tonien, K. Kurosawa, and K. Toyota. Birthday paradox for multi-collisions. In M. S. Rhee and B. Lee, editors, *Information Security and Cryptology - ICISC 2006, 9th International Conference, Busan, Korea, November 30 - December 1, 2006, Proceedings*, volume 4296 of *LNCS*, pages 29–40. Springer, 2006.
36. L. Zhang, W. Wu, P. Wang, L. Zhang, S. Wu, and B. Liang. Constructing rate-1 macs from related-key unpredictable block ciphers: PGV model revisited. In *FSE*, volume 6147 of *LNCS*, pages 250–269. Springer, 2010.

A Proof details for LR-MAC1.

Here, we present the details of the proof of Thm. 1.

The t_1-CR-*adversary* B. B has to find a collision for the hash function H_s. At the start of the game, she is provided a key s for the hash function, which she relays to A^{L}. Then, she picks a key k uniformly at random for F. Moreover, she has a list $\mathcal{H}$ which is empty at the start.
When A does a tag-generation query on input m^i, B (1) simply computes $h^i = \mathsf{H}_s(m^i)$ and she adds $\{(m^i, h^i)\}$ to $\mathcal{H}$, then, (2) she computes $\tau^i = \mathsf{F}_k^{h^i}(0^n)$ and collects the leakage $\mathsf{l}_{\mathsf{e}}^i = \mathsf{L}_{\mathsf{F}}(h^i, 0^n; k)$ Finally (3), she answers τ^i to A and the leakage $L_M^i = \mathsf{l}_{\mathsf{e}}^i$ and she adds $\{(m^i, h^i)\}$ to $\mathcal{H}$.
This takes time $t_{\mathsf{H}} + t_{\mathsf{F}} + t_{\mathsf{L}(\mathsf{F})}$.
When A asks a verification query on input (m^i, τ^i), B first, (1) she computes $h^i = \mathsf{H}_s(m^i)$, adds $\{(m^i, h^i)\}$ to $\mathcal{H}$, then (2), she computes $\tilde{x}^i = \mathsf{F}_k^{h^i,-1}(\tau^i)$ and collects the leakage $\mathsf{l}_{\mathsf{i}}^i = \mathsf{L}_{\mathsf{F}^{-1}}(h^i, \tau^i; k)$. Finally (3), if $0^n = \tilde{x}^i$, B answers $\top$ to A; otherwise, $\bot$. Moreover, B gives the leakage $L_V^i = (\tilde{x}^i, \mathsf{l}_{\mathsf{i}}^i)$ to A.
This takes time $t_{\mathsf{H}} + t_{\mathsf{F}} + t_{\mathsf{L}(\mathsf{F})}$.
When A outputs its forgery $(m^{q_V+1}, \tau^{q_V+1})$, B computes $h^{q_V+1} = \mathsf{H}_s(m^{q_V+1})$, adds $\{(m^{q_V+1}, h^{q_V+1})\}$ to $\mathcal{H}$. This takes time t_{H}.
At the end of the game, B looks up the list $\mathcal{H}$ to find a collision. If she finds it, she outputs it; otherwise, 0^n and 1^n.
Thus, in total, she needs time $t + (q_M + q_V + 1)t_{\mathsf{H}} + (q_M + q_V)(t_{\mathsf{F}} + t_{\mathsf{L}(\mathsf{F})})$. [13]

The $(q_{\mathsf{L}}, q_{\mathsf{Eval}}, q_{\mathsf{Inv}}, t_2)$-$\mathsf{sUP\text{-}L2}$-*adversary* C^j. C^i has to find a valid triple (x, tw, y) for F_k which is fresh and valid.
At the start of the game, she picks uniformly at random a key $s \xleftarrow{\$} \mathcal{HK}$ for the hash function, which she relays to A^{L}. Moreover, she has a list $\mathcal{S}$, which is empty at the start of the game.
When A^{L} does a query to L on input $(tw, x; k')$ either to Eval or to Inv, C^j does

[13] Since we assume that A has access to $\mathsf{L}(\cdot)$, we do not need to consider what B does with these queries. A does them whenever A wants and she does them in his running time.

the same query. She receives the leakage L, which she forwards to A.
When A does a tag-generation query on input m^i, C^j (1) simply computes $h^i = \mathsf{H}_s(m^i)$, then, (2) she queries his oracle LEval on input $(h^i, 0^n)$ receiving τ^i and collects the leakage l_e^i, finally (3), she answers A τ^i and the leakage $L_M^i = \mathsf{l}_\mathsf{e}^i$; moreover, she adds $\{(m^i, \tau^i)\}$ to $\mathcal{S}$.
This takes time t_H.
When A asks one of the first j verification query on input (m^i, τ^i), C^j for the first, (1) she computes $h^i = \mathsf{H}_s(m^i)$, then (2), she queries his oracle LInv on input (τ^i, h^i) receiving $\tilde{x}^i$ and collects the leakage l_i^i. Finally (3), if $0^n = \tilde{h}^i$ and $\{(m^i, \tau^i)\} \in \mathcal{S}$ (3a), C^j answers $\top$ and the leakage $L_V^i = (\tilde{x}^i, \mathsf{l}_\mathsf{i}^i)$ to A; if $0^n = \tilde{h}^i$ and $\{(m^i, \tau^i)\} \notin \mathcal{S}$ (3b), C^j aborts; otherwise (3c), $\perp$. Moreover, C^j gives the leakage $L_V^i = (\tilde{x}^i, \mathsf{l}_\mathsf{i}^i)$ to A.
This takes time t_H.
When A asks his j+1-th verification query on input $(m^{q_V+1}, \tau^{q_V+1})$, C^j computes (1) $h^{j+1} = \mathsf{H}_s(m^{j+1})$, and (2) she outputs $(0^n, h^{j+1}, \tau^{j+1})$.
This takes time t_H.
Thus, in total, C^i does at most q_L query to L, q_M to LEval and $j \leq q_v$ to LInv; moreover, she needs time at most $t + (q_M + j + 1)t_\mathsf{H} \leq t_2$.

B Additional figures and algorithms

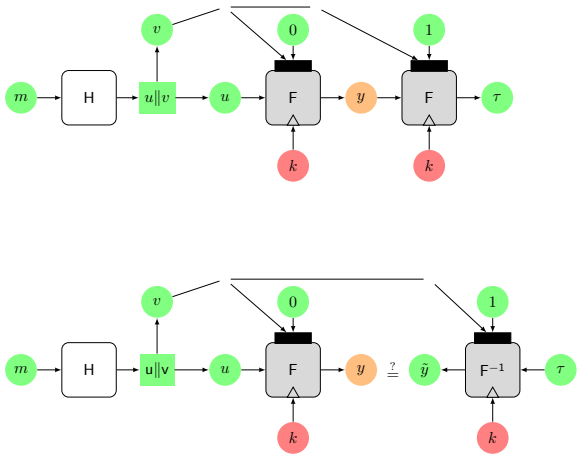

Fig. 4: LR-MAC2 variant with a single key.

Algorithm 4 LR-MAC2 algorithm with a single key key .

It uses a strongly protected TBC $\mathsf{F} : \mathcal{K} \times \mathcal{TW} \times \{0,1\}^n \to \{0,1\}^n$ and a hash function $\mathsf{H} : \mathcal{HK} \times \{0,1\}^n \to \{0,1\}^{2n}$.

- Gen
  - $k \xleftarrow{\$} \mathcal{K}^2$
  - $s \xleftarrow{\$} \mathcal{HK}$
- $\mathsf{Mac}_k(m)$:
  - $u \| v \xleftarrow{n} \mathsf{H}_s(m)$
  - $y = \mathsf{F}_k^{v\|0}(u)$
  - $\tau = \mathsf{F}_k^{v\|1}(y)$
  - Return τ
- $\mathsf{Vrfy}_k(m, \tau)$:
  - $u \| v \xleftarrow{n} \mathsf{H}_s(m)$
  - $y = \mathsf{F}_k^{v\|0}(u)$
  - $\tilde{y} = \mathsf{F}_k^{v\|1,-1}(\tau)$
  - If $y == \tilde{y}$ Return $\top$
  - Else Return $\perp$

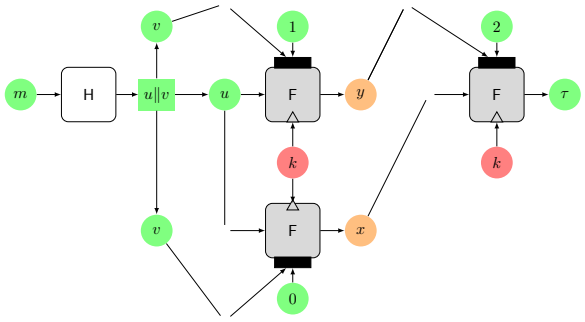

Fig. 5: LR-MAC2 variant with 3 TBC calls.

C Proof for LR-MAC2.

Here, we present the details of the proof of Thm. 2.

Proof. To prove the theorem, we follow a sequence of games.
Game 0. Let Game 0 be the sUF−L2 game where the $(q_{\mathsf{L}}, q_M, q_V, t)$-adversary A^{L} tries to produce a forgery when she plays against LR-MAC2. Let E_0 be the event that the adversary wins the game; that is, the output of the game is 1.

Game 1. Game 1 is Game 0, except that we abort if there is a full collision in the hash function. Let E_1 be the event that the adversary wins the game; that

is, the output of the game is 1. In a similar vein to the proofs before, it's easy to show

$$|\Pr[E_0] - \Pr[E_1]| \leq \epsilon_{\mathsf{CR}(2n)}.$$

To do this, we use a t_1 adversary BB.

The t_1-CR-adversary BB. BB has to find a collision for the hash function H_s. At the start of the game, she is provided a key s for the hash function, which he relays to A^{L}. Then, she picks two key k_1, k_2 uniformly at random for F. Moreover, she has a list $\mathcal{H}$ which is empty at the start.

When A does a tag-generation query on input m^i, B (1) simply computes $u^i \| v^i = h^i = \mathsf{H}_s(m^i)$ and she adds $\{(m^i, h^i)\}$ to $\mathcal{H}$, then, (2) she computes $y^i = \mathsf{F}_{k_1}^{v^i}(u^i)$ and collects the leakage $\mathsf{l}_{\mathsf{e}}^{i,0} = \mathsf{L}_{\mathsf{F}}(v^i, u^i; k_1)$, after that, (3) she computes $\tau^i = \mathsf{F}_{k_2}^{v^i}(y^i)$ and collects the leakage $\mathsf{l}_{\mathsf{e}}^{i,1} = \mathsf{L}_{\mathsf{F}}(v^i, y^i; k_2)$ Finally (4), she answers τ^i to A and the leakage $L_M^i = (y^i, \mathsf{l}_{\mathsf{e}}^{i,0}, \mathsf{l}_{\mathsf{e}}^{i,1})$ and she adds $\{(m^i, h^i)\}$ to $\mathcal{H}$.

This takes time $t_{\mathsf{H}} + 2t_{\mathsf{F}} + 2t_{\mathsf{L}(\mathsf{F})}$.

When A asks a verification query on input (m^i, τ^i), B first, (1) she computes $u^i \| v^i = h^i = \mathsf{H}_s(m^i)$, adds $\{(m^i, h^i)\}$ to $\mathcal{H}$, then, (2) she computes $y^i = \mathsf{F}_{k_1}^{v^i}(u^i)$ and collects the leakage $\mathsf{l}_{\mathsf{e}}^{i,0} = \mathsf{L}_{\mathsf{F}}(v^i, u^i; k_1)$, after that (3), she computes $\tilde{y}^i = \mathsf{F}_{k_2}^{v^i,-1}(\tau^i)$ and collects the leakage $\mathsf{l}_{\mathsf{i}}^{i,1} = \mathsf{L}_{\mathsf{F}^{-1}}(v^i, \tau^i; k_2)$. Finally (4), if $y^i = \tilde{y}^i$, B answers $\top$ to A; otherwise, $\perp$. Moreover, B gives the leakage $L_V^i = (y^i, \tilde{y}^i, \mathsf{l}_{\mathsf{e}}^{i,0}, \mathsf{l}_{\mathsf{i}}^{i,1})$ to A.

This takes time $t_{\mathsf{H}} + 2t_{\mathsf{F}} + 2t_{\mathsf{L}(\mathsf{F})}$.

When A outputs its forgery $(m^{q_V+1}, \tau^{q_V+1})$, B computes $h^{q_V+1} = \mathsf{H}_s(m^{q_V+1})$, adds $\{(m^{q_V+1}, h^{q_V+1})\}$ to $\mathcal{H}$. This takes time t_{H}.

At the end of the game, B looks up the list $\mathcal{H}$ to find a collision. If she finds it, she outputs it; otherwise, 0 and 1.

Thus, in total, she needs time $t + (q_M + q_V + 1)t_{\mathsf{H}} + 2(q_M + q_V)(t_{\mathsf{F}} + t_{\mathsf{L}(\mathsf{F})})$. [14]

Game 2. Game 2 is Game 1, except that we abort if there are at least μ half collisions on the same value v for the hash function. That is, there μ hash collisions for H_s^2.

Let E_2 be the event that the adversary wins the game; that is, the output of the game is 1.

Transition between Game 1 and Game 2. Clearly, Game 1 and Game 2 are identical if the following event μ-HHC (μ- *half hash collision*) does not happen:

$$\mu\text{-}HHC := \left\{ \begin{array}{l} \exists i_1, ..., i_\mu \in \{1, ..., q_M\} \cup \{1, ..., q_V + 1\} \\ \text{ with } \forall j, l \in 1, ..., \mu \; i_j \overset{\%}{\neq} i_l \text{ s.t. } v^{i_j} = v^{i_l} \text{ and } m^{i_j} \neq m^{i_l} \end{array} \right\}.$$ [15]

[14] Since we assume that A has access to $\mathsf{L}(\cdot)$, we do not need to consider what she does with these queries. A does them whenever she wants and she does them in his running time.

[15] $i_j \overset{\%}{\neq} i_l$ means that if i_j comes from a tag-generation query and i_l from a verification query, or viceversa, then, they are considered differently.

To compute this event, we build a t_1-μ-CR-adversary B.

The t_1-l-CR-adversary B. B has to find l half collisions for the hash function H_s (on the last half). At the start of the game, she is provided a key s for the hash function, which he relays to A^{L}. Then, she picks two keys k_1, k_2 uniformly at random for F. Moreover, she has a list $\mathcal{H}$ which is empty at the start.
When A does a tag-generation query on input m^i, B (1) simply computes $u^i \| v^i = h^i = \mathsf{H}_s(m^i)$ and she adds $\{(m^i, h^i)\}$ to $\mathcal{H}$, then, (2) she computes $y^i = \mathsf{F}_{k_1}^{v^i}(u^i)$ and collects the leakage $\mathsf{l}_{\mathsf{e}}^{i,0} = \mathsf{L}_{\mathsf{F}}(v^i, u^i; k_1)$, after that, (3) she computes $\tau^i = \mathsf{F}_{k_2}^{v^i}(y^i)$ and collects the leakage $\mathsf{l}_{\mathsf{e}}^{i,1} = \mathsf{L}_{\mathsf{F}}(v^i, y^i; k_2)$ Finally (4), she answers τ^i to A and the leakage $L_M^i = (y^i, \mathsf{l}_{\mathsf{e}}^{i,0}, \mathsf{l}_{\mathsf{e}}^{i,1})$ and she adds $\{(m^i, h^i)\}$ to $\mathcal{H}$.
This takes time $t_{\mathsf{H}} + 2t_{\mathsf{F}} + 2t_{\mathsf{L}(\mathsf{F})}$.
When A asks a verification query on input (m^i, τ^i), B first, (1) she computes $u^i \| v^i = h^i = \mathsf{H}_s(m^i)$, adds $\{(m^i, h^i)\}$ to $\mathcal{H}$, then, (2) she computes $y^i = \mathsf{F}_{k_1}^{v^i}(u^i)$ and collects the leakage $\mathsf{l}_{\mathsf{e}}^{i,0} = \mathsf{L}_{\mathsf{F}}(v^i, u^i; k_1)$, after that (3), she computes $\tilde{y}^i = \mathsf{F}_{k_2}^{v^i,-1}(\tau^i)$ and collects the leakage $\mathsf{l}_{\mathsf{i}}^{i,1} = \mathsf{L}_{\mathsf{F}^{-1}}(v^i, \tau^i; k_2)$. Finally (4), if $y^i = \tilde{y}^i$, B answers $\top$ to A; otherwise, $\bot$. Moreover, B gives the leakage $L_V^i = (y^i, \tilde{y}^i, \mathsf{l}_{\mathsf{e}}^{i,0}, \mathsf{l}_{\mathsf{i}}^{i,1})$ to A.
This takes time $t_{\mathsf{H}} + 2t_{\mathsf{F}} + 2t_{\mathsf{L}(\mathsf{F})}$.
When A outputs its forgery $(m^{q_V+1}, \tau^{q_V+1})$, B computes $h^{q_V+1} = \mathsf{H}_s(m^{q_V+1})$, adds $\{(m^{q_V+1}, h^{q_V+1})\}$ to $\mathcal{H}$. This takes time t_{H}.
At the end of the game, B looks up the list $\mathcal{H}$ to l half collisions on v. If she finds it, she outputs it; otherwise, $\{0, 1, ..., (l-1)_2\}$, where with $(N-1)_2$ we denote the binary representation of $N-1$.
Thus, in total, she needs time $t + (q_M + q_V + 1)t_{\mathsf{H}} + 2(q_M + q_V)(t_{\mathsf{F}} + t_{\mathsf{L}(\mathsf{F})}) \leq t_1$. [16]

Bounding $|\Pr[E_1] - \Pr[E_2]|$. Observe that if event μ-HHC happens, B wins. Note that B simulates correctly Game 1 for A^{L}.
Since B is a t_1-adversary and H is a $(t_1, \epsilon_{\mu\text{-}\mathsf{CR}(n)})$-collision resistant hash function, we can bound:

$$|\Pr[E_1] - \Pr[E_2]| \leq \epsilon_{\mu\text{-}\mathsf{CR}(n)}.$$

Game 3 Game 3 is Game 2, except that we abort if there is a query to F_{k_1} which can be used straightforwardly as a forgery. More formally, we want to avoid that when we compute for the first time the y^i associated to a message m^i, we obtain that y^i is equal to a $\tilde{y}^j$ already defined in the verification query (m^j, τ^j) where $v^j = v^i$ (remind that $\mathsf{H}(m) = h = u \| v$).
We call this event FfD (which stands forgeries from direct query to F_{k_1}). For-

[16] Since we assume that A has access to $\mathsf{L}(\cdot)$, we do not need to consider what she does with these queries. A does them whenever she wants and she does them in his running time.

mally,

$$FfD := \left\{ \begin{array}{l} \quad \exists i \in \{1,...,q_V+1\} \text{ s.t. } \exists j \in \{1,...,i\} \\ \quad \text{s.t. } m^i \neq m^j,\ v^i = v^j,\ y^i = \tilde{y}^j \\ \wedge\ \forall l \in \{1,...,q_M\}\ m^l \neq m^i, \\ \wedge\ \forall I \in \{1,...,i\} \subset \{1,...,q_{V+1}\}\ m^i \neq m^I \end{array} \right\}$$

and we abort if event FfD happens [17]. Let E_3 be the event that the adversary wins the game; that is, the output of the game is 1.

Transition between Game 2 ad Game 3. We build a sequence of $q_V + 2$ games: Game 2^0,...,Game 1^{q_V+1}.

Game 2^j. Game 2^j is Game 2, where we abort if one of the first j verification queries has triggered event FfD. Thus, Game 2^0 is Game 2, while Game 2^{q_V+1} is Game 3. We denote with FfD_j the fact that after j verification queries event FfD has been triggered.
Let E_2^j be the event that the adversary wins the game.

Transition between Game 2^j and Game 2^{j+1}. Clearly, Game 2^j and Game 2^{j+1} are identical if the $j+1$-th verification query does not trigger event FfD. Clearly, $\left|\Pr[E_2^j] - \Pr[E_2^{j+1}]\right| \leq \Pr[FfD_{j+1}|\neg FfD_j]$. To bound the probability that event $FfD_{j+1}|\neg FfD_j$ happens we build a $(q_{\mathsf{LF}}, q_{\mathsf{Eval}}, q_{\mathsf{Inv}}, t_2)$-sUP-L2-adversary C^j against F.

The $(q_{\mathsf{LF}}, q_{\mathsf{Eval}}, q_{\mathsf{Inv}}, t_2)$-sUP-L2-adversary C^j. C^i has to find a valid triple (tw, x, y) for F_{k_1} which is fresh and valid.
At the start of the game, she picks uniformly at random a key $s \xleftarrow{\$} \mathcal{HK}$ for the hash function, which she relays to A^{L}. In addition, she picks uniformly at random a key k_2. Moreover, she has three lists $\mathcal{S}$ (to keep tag-generation queries), $\mathcal{DC}$ (to keep the couples (u, v, y) of verification queries) and $\mathcal{IT}$ (to keep the triples $(\tilde{y}, v, \tau)$ of verification queries), which are empty at the start of the game.
When A^{L} does a query to L on input $(tw, x; k')$ either to Eval or to Inv, D^j does the same query. She receives the leakage L, which she forwards to A. [18]
When A does a tag-generation query on input m^i, C^j (1) simply computes $u^i \| v^i = h^i = \mathsf{H}_s(m^i)$, then, (2) she calls her oracle on input (v^i, u^i) obtaining y^i and $\mathsf{l}_{\mathsf{e}}^{i,0}$, after that (3), she computes $\tau^i = \mathsf{F}_{k_2}^{v^i}(y^i)$ and collects the

[17] We do not consider the case that event FfD is triggered by a tag-generation queries, that is the y^i defined during the ith tag-generation query on input m^i is equal to the $\tilde{y}^j$ defined during the jth verification query with $v^i = v^j$. This for two reasons. First, we observe that if $m^i = m^j$, we cannot create a forgery from m^i, since the forgery attempt will not be fresh. Second, we observe that to force to obtain y^i during a verification query on input m^j with $v^j = v^i$, we observe that we must have $u^i = u^j$, thus, we need a full collision on the hash value h.

[18] For simplicity, we suppose that A tries only to model the leakage of F and not the leakage of the full MAC. In fact, she can compute everything. Anyway the extension is straightforward.

leakage $\mathsf{l}_{\mathsf{e}}^{i,1} = \mathsf{L}_{\mathsf{LEval}}(v^i, y^i; k_2)$, finally (4), she answers A τ^i and the leakage $L_M^i = (y^i, \mathsf{l}_{\mathsf{e}}^{i,0}, \mathsf{l}_{\mathsf{e}}^{i,1})$; moreover, she adds $\{(m^i, \tau^i)\}$ to $\mathcal{S}$ and $\{(u^i, v^i \| 0, y^i)\}$ to $\mathcal{DC}$.
This takes time $t_{\mathsf{H}} + t_{\mathsf{F}} + t_{\mathsf{L(F)}}$ and she does one query to LEval.
When A asks one of the first j verification query on input (m^i, τ^i), D^j first (1), computes $u^i \| v^i = h^i = \mathsf{H}_s(m^i)$, 1) simply computes $u^i \| v^i = h^i = \mathsf{H}_s(m^i)$, then, (2) she calls her oracle on input (v^i, u^i) obtaining y^i and $\mathsf{l}_{\mathsf{e}}^{i,0}$, after that (3), she computes $\tilde{y}^i = \mathsf{F}_{k_2}^{v^i,-1}(\tau^i)$ and collects the leakage $\mathsf{l}_{\mathsf{i}}^{i,1} = \mathsf{L}_{\mathsf{Inv}}(v^i, \tau^i; k_2)$, Finally (4), if $y^i = \tilde{y}^i$ and $\{(m^i, \tau^i)\} \in \mathcal{S}$ (4a), C^j answers $\top$ and the leakage $L_V^i = (y^i, \tilde{y}^i, \mathsf{l}_{\mathsf{e}}^{i,0}, \mathsf{l}_{\mathsf{i}}^{i,1})$ to A; if $y^i = \tilde{y}^i$ and $\{(m^i, \tau^i)\} \notin \mathcal{S}$ (4b), C^j aborts; otherwise (4c), $\perp$. Moreover, C^j gives the leakage $L_V^i = (y^i, \tilde{y}^i, \mathsf{l}_{\mathsf{e}}^{i,0}, \mathsf{l}_{\mathsf{i}}^{i,1})$ to A.
Then, she adds $\{(v^i, u^i, y^i)\}$ to $\mathcal{DC}$ and $\{(v^i, \tilde{y}^i, \tau^i)\}$ to $\mathcal{IT}$.
This takes time $t_{\mathsf{H}} + t_{\mathsf{F}} + t_{\mathsf{L(F)}}$. Moreover, C^j does one query to LInv.
When A asks his $j+1$-th verification query on input $(m^{q_V+1}, \tau^{q_V+1})$, D^j computes (1) $u^{j+1} \| v^{j+1} = h^{j+1} = \mathsf{H}_s(m^{j+1})$, then (2), if (2a) there is a query on input $(v^{j+1}, u^{j+1}, y) \in \mathcal{DC}$ for any y, she proceeds as for the previous verification queries; otherwise (2c), C^j defines the set T_j (target for the adversary C^j) as follow:

$$T_j := \{(v, \tilde{y}, \tau) \in \mathcal{IT} \text{ s.t. } v = v^{j+1}\}$$

Then (3c), she picks uniformly at random an element $(v^{j+1}, \tilde{y}^*, \tau^*)$ in T_j, and she outputs $(v^{j+1}, u^{j+1}, \tilde{y}^*)$. This takes time t_{H}.
Thus, in total, C^i does at most q_{L} query to L, $q_M + q_V$ to LEval and $j \leq q_V$ to LInv; moreover, she needs time at most $t + (q_M + j + 1)t_{\mathsf{H}} + (q_M + q_V)(t_{\mathsf{F}} + t_{\mathsf{L(F)}} \leq t_2$.
Bounding $|\Pr[E_2^j] - \Pr[E_2^{j+1}]|$**.** Note that C^j simulates correctly Game 2^j for A^{L}.
We observe that if event $Ffc_{j+1} | \neg Ffc_j$, the adversary C^j does a correct guess if she has picked correctly the target $\tilde{y}^*$. This happens with probability $|T_j|^{-1}$.
Since C^j is a $(q_{\mathsf{L}}, q_{\mathsf{LEval}}, q_{\mathsf{LInv}}, t_2)$-adversaries and F is a $(q_{\mathsf{L}}, q_{\mathsf{LEval}}, q_{\mathsf{LInv}}, t_2, \epsilon_{\mathsf{sUP\text{-}L2}})$-strongly unpredictable with leakage, we can bound:

$$|\Pr[E_2^j] - \Pr[E_2^{j+1}]| \leq \Pr[FfD_{j+1} | \neg FfD_j] \leq \epsilon_{\mathsf{sUP\text{-}L2}} |T_j|.$$

Bounding $|\Pr[E_2] - \Pr[E_3]|$**.** Iterating, we obtain

$$|\Pr[E_2] - \Pr[E_3]| \leq \sum_{j=0}^{q_V+1} |\Pr[E_2^j] - \Pr[E_2^{j+1}]| \leq \sum_{j=0}^{q_V+1} |T_j| \epsilon_{\mathsf{sUP\text{-}L2}} = \epsilon_{\mathsf{sUP\text{-}L2}} \sum_{j=0}^{q_V+1} |T_j|.$$

So, now we have to only to bound $\sum_{j=0}^{q_V+1} |T_j|$.

Bounding $\sum_{j=0}^{q_V+1} |T_j|$**.** The simplest way to bound it is to say that $T_j \leq j$, then to use the well known result from Gauss $\sum_{j=0}^{q_V} j = \frac{(q_V+1)(q_V+2)}{2}$.

But, we want to be more precise. We start defining $T = \bigcup_{j=0}^{q_V+1} T_j$ and observing

that $|T| = q_V + 1$. Moreover $T_j \cap T_{j'} = \emptyset$ if $v^{j+1} \neq v^{j'+1}$.
Then, let $\{v_1, ..., v_I\}$ be the tweaks used in the LInv queries used in decryption queries (thus, $v_1, ..., v_I$ are the vs defined during the verification queries). Clearly $I \leq q_V + 1$.
Now, we define $T_{v_i} := \bigcup_{\substack{l=1 \\ v^l = v_i}}^{q_V+1} T_l$.[19] That is, T_{v_i} is the set of all triples $(v_l, \tilde{y}, \tau)$ defined during verification queries. Moreover, for every v^i used in a verification query, we define $T(v^i) := T_{v^i}$. Note that $\forall l$, $T_l \subset T_{v^l}$ (in T_{v^l} we have more targets than in T_l, the ones obtained by subsequent verification queries).
Moreover, clearly $T = \bigcup_{l=1}^{I} T_{v_l}$ and $\forall l, l' = 1, ..., I$ $T_{v_l} \cap T_{v_{l'}} = \emptyset$ iff. $l \neq l'$.
Thus, we can bound $\sum_{j=0}^{q_V+1} |T_j| \leq \sum_{j=0}^{q_V+1} |T_{v_j}|$.
Let $W(v^i) := \{m$ used in verification queries s.t. $\mathsf{H}_s(m) = u \| v^i$ for any $u\}$.
Thus, $\sum_{j=0}^{q_V+1} |T_{v^j}| = \sum_{j=0}^{I} |W(v_j)||T_{v_j}|$. Note that if the same m is used in two different verification queries, we put m only once in $W(v)$ with $u \| v = \mathsf{H}_s(m)$.
Clearly $\forall j = 1, ..., I$, $|W(v^i)| \leq \mu$. In fact, we are in Game 3, where event $\mu\text{-}HHC$ has not happened. Moreover, $\sum_{j=1}^{I} |W(v_j)| = q_V + 1$. Thus,

$$\sum_{j=0}^{I} |W(v_j)| \cdot |T_{v_j}| \leq \mu \sum_{j=0}^{I} |T_{v_j}| = \mu |T| = \mu(q_V + 1).$$

Where the second to last equality comes from the fact T is the union of the T_{v_j}s which are pairwise disjoint.

Computing $|\Pr[E_2] - \Pr[E_3]|$. Putting the previous 2 paragraphs together, we obtain that

$$|\Pr[E_2] - \Pr[E_3]| \leq \epsilon_{\mathsf{sUP\text{-}L2}} \sum_{j=0}^{q_V} |T_j| \leq \mu(q_V + 1) \cdot \epsilon_{\mathsf{sUP\text{-}L2}}.$$

Game 4 Game 4 is Game 3, except that we abort if there is a backward query to F_{k_2} which provokes a forgery. More formally, we want to avoid that when we compute for the first time the $\tilde{y}^i$ associated to a tag τ^i and a tweak v^i, we obtain that $\tilde{y}^i$ is equal to a y already defined in the verification query (m^j, τ^j) where $v^j = v^i$ (remind that $\mathsf{H}(m) = h = u \| v$).
We call this event FfI (which stands forgeries from inverse query to F_{k_2}). For-

[19] Note that with v^l we denote v obtained in the lth verification query. Instead with v_l we denote the lth element in the previous list, $\{v_1, ..., v_I\}$.

mally,

$$FfI := \left\{ \begin{array}{l} \quad \exists i \in \{1,...,q_V\} \text{ s.t. } \exists j \in \{1,...,i-1\} \\ \quad \text{s.t. } m^i \neq m^j,\ v^i = v^j,\ \tilde{y}^i = y^j \\ \wedge\ \forall l \in \{1,...,q_M\}\ m^l \neq m^i, \\ \wedge\ \forall I \in \{1,...,i-1\} \subset \{1,...,q_{V+1}\}\ m^i \neq m^I \end{array} \right\}$$

and we abort if event FfI happens [20]. Let E_4 be the event that the adversary wins the game; that is, the output of the game is 1.

Transition between Game 3 and Game 4. We build a sequence of $q_V + 1$ games: Game 3^0,...,Game 3^{q_V}.

Game 3^j. Game 3^j is Game 3, where we abort if one of the first j verification queries has triggered event FfI. Thus, Game 3^0 is Game 3, while Game 3^{q_V} is Game 4. We denote with FfI_j the fact that after j verification queries event FfI has been triggered.
Let E_3^j be the event that the adversary wins this game.

Transition between Game 3^j and Game 3^{j+1}. Clearly, Game 3^j and Game 3^{j+1} are identical if the $j+1$-th tag-generation query does not trigger event FfD. Clearly, $\left|\Pr[E_3^j] - \Pr[E_3^{j+1}]\right| \leq \Pr[FfI_{j+1}|\neg FfI_j]$. To bound the probability that event $FfI_{j+1}|\neg FfI_j$ happens we build a $(q_{\mathsf{LF}}, q_{\mathsf{Eval}}, q_{\mathsf{Inv}}, t_2)$-sUP−L2-adversary D^j against F.

The $(q_{\mathsf{LF}}, q_{\mathsf{Eval}}, q_{\mathsf{Inv}}, t_2)$-sUP−L2-*adversary* D^j. D^i has to find a valid triple (x, tw, y) for F_{k_1} which is fresh and valid.
At the start of the game, she picks uniformly at random a key $s \xleftarrow{\$} \mathcal{HK}$ for the hash function, which she relays to A^{L}. in addition, she picks a key k_2 uniformly at random. Moreover, she has three lists $\mathcal{S}$ (to keep tag-generation queries), $\mathcal{DT}$ (to keep the couples $(u, v\|0, y)$ of verification queries) and $\mathcal{IQ}$ (to keep the triples $(\tilde{y}, v\|1, \tau)$ of verification queries), which are empty at the start of the game.
When A^{L} does a query to L on input $(tw, x; k')$ either to Eval or to Inv, EE^j does the same query. She receives the leakage L, which she forwards to A. [21]
When A does a tag-generation query on input m^i, EE^j (1) simply computes

[20] We do not consider the case that event FfI is triggered by the equality of $\tilde{y}$ to a y defined in a tag-generation query, that is if the $\tilde{y}^i$ defined during the ith verification query on input m^i is equal to the y^j defined during the jth tag-generation query with $v^i = v^j$, because this cannot result in a forgery for m^j, since that forgery attempt will not be fresh. Moreover, we observe that to force to obtain $\tilde{y}^j = y^i$ during a verification query on input m^j with $v^j = v^i$, we observe that we must have $u^i = u^j$, thus, we need a full collision on the hash value h with m^j.

[21] For simplicity, we suppose that A tries only to model the leakage of F and not the leakage of the full MAC. In fact, she can compute everything. Anyway the extension is straightforward.

$u^i\|v^i = h^i = \mathsf{H}_s(m^i)$, then, (2) she computes $\mathsf{F}_{k_1}^{v^i}(u^i)$ and computes the leakage $\mathsf{l}_{\mathsf{e}}^{i,0} = \mathsf{L}_{\mathsf{Eval}}(v^i, u^i; k_1)$, after that (3), she queries his oracle on input (v^i, y^i) receiving τ^i and collects the leakage $\mathsf{l}_{\mathsf{e}}^{i,1}$, finally (4), she answers A τ^i and the leakage $L_M^i = (y^i, \mathsf{l}_{\mathsf{e}}^{i,0}, \mathsf{l}_{\mathsf{e}}^{i,1})$; moreover, she adds $\{(m^i, \tau^i)\}$ to $\mathcal{S}$.
This takes time $t_{\mathsf{F}} + t_{\mathsf{L}(\mathsf{F})}$ and she does one query to LEval.
When A asks one of the first j verification query on input (m^i, τ^i), EE^j first, (1) computes $u^i\|v^i = h^i = \mathsf{H}_s(m^i)$, then, (2) she computes $\mathsf{F}_{k_1}^{v^i}(u^i)$ and computes the leakage $\mathsf{l}_{\mathsf{e}}^{i,0} = \mathsf{L}_{\mathsf{Eval}}(v^i, u^i; k_1)$, after that (3), she queries his oracle LInv on input (v^i, τ^i) receiving $\tilde{y}^i$ and collects the leakage $\mathsf{l}_{\mathsf{i}}^{i,1}$. Finally (4), if $y^i = \tilde{y}^i$ and $\{(m^i, \tau^i)\} \in \mathcal{S}$ (4a), D^j answers $\top$ and the leakage $L_V^i = (y^i, \tilde{y}^i, \mathsf{l}_{\mathsf{e}}^{i,0}, \mathsf{l}_{\mathsf{i}}^{i,1})$ to A; if $y^i = \tilde{y}^i$ and $\{(m^i, \tau^i)\} \notin \mathcal{S}$ (4b), EE^j aborts; otherwise (4c), $\perp$. Moreover, D^j gives the leakage $L_V^i = (y^i, \tilde{y}^i, \mathsf{l}_{\mathsf{e}}^{i,0}, \mathsf{l}_{\mathsf{i}}^{i,1})$ to A.
Then, she adds $\{(u^i, v^i\|0, y^i)\}$ to $\mathcal{DT}$ and $\{(\tilde{y}^i, v^i\|1, \tau^i)\}$ to $\mathcal{IQ}$.
This takes time $t_{\mathsf{H}} + t_{\mathsf{F}} + t_{\mathsf{L}(\mathsf{F})}$. Moreover, D^j does one query to $lf\mathsf{Inv}$.
When A asks his $j+1$-th verification query on input $(m^{q_V+1}, \tau^{q_V+1})$, D^j computes (1) $u^{j+1}\|v^{j+1} = h^{j+1} = \mathsf{H}_s(m^{j+1})$, then (2), if (2a) there is a query on input $(v^{j+1}, \tilde{y}, \tau^{j+1})\} \in \mathcal{DC}$ for any $\tilde{y}$, she proceeds as for the previous verification queries; otherwise (2b), D^j defines the set IT_j (target for the adversary D^j) as follow:

$$IT_j := \{(v, u, y) \in \mathcal{IT} \text{ s.t. } v = v^{j+1}\}$$

Then (3c), she picks uniformly at random an element (v^{j+1}, u^*, y^*) in IT_j, and she outputs $(v^{j+1}, y^*, \tau^{j+1})$ as her prediction. This takes time t_{H}.
Thus, in total, D^i does at most q_{L} query to L, $q_M \leq q_M$ to LEval and $j \leq q_V$ to LInv; moreover, she needs time at most $t + (q_M + j + 1)t_{\mathsf{H}}(t_{\mathsf{F}} + t_{\mathsf{L}(\mathsf{F})})(q_M + q_V) \leq t_2$.

Bounding $|\Pr[E_3^j] - \Pr[E_3^{j+1}]|$**.** Note that D^i simulates correctly Game 1^j for A^{L}.
We observe that if event $FfI_{j+1}|\neg FfI_j$, the adversary D^j does a correct guess if she has picked correctly the target $\tilde{y}^*$. This happens with probability $|IT_j|^{-1}$. Since D^j is a $(q_{\mathsf{L}}, q_{\mathsf{Eval}}, q_{\mathsf{Inv}}, t_2)$-adversaries and F is a $(q_{\mathsf{L}}, q_{\mathsf{Eval}}, q_{\mathsf{Inv}}, t_2, \epsilon_{\mathsf{sUP\text{-}L2}})$-strongly unpredictable with leakage, we can bound:

$$|\Pr[E_3^j] - \Pr[E_3^{j+1}]| \leq \Pr[FfI^{j+1}|\neg FfI_j] \leq \epsilon_{\mathsf{sUP\text{-}L2}}|IT_j|.$$

Bounding $\sum_{j=0}^{q_V} |IT_j|$**.** As we did for $\sum_{j=0}^{q_V+1} T_j$, the simplest way to bound it is to say that $IT_j \leq j$, then to use the well known result from Gauss $\sum_{j=0}^{q_V} j = \frac{q_V(q_V+1)}{2}$.
But, as there we want to be more precise. We start observing that $|IT| = q_V - 1$ with $IT = \bigcup_{j=0}^{q_V} IT_j$. Moreover $IT_j \cap IT_{j'} = \emptyset$ if $v^{j+1} \neq v^{j'+1}$.
Then, let $v_1\|1, ..., v_I\|1$ be the tweaks used in the LInv queries used in decryption queries (thus, $v_1, ..., v_I$ are the vs computed during the verification queries). [22]

[22] As there, note that with v^l we denote v obtained in the lth verification query. Instead with v_l we denote the lth element in the previous list, $\{v_1\|1, ..., v_I\|1$.

Clearly $I \leq q_V$.

Now, we define $IT_{v_i} := \bigcup_{\substack{l=1 \\ v^l = v_1}}^{q_V} IT_l$. Moreover, for every v^l used in a verification query, we define $IT(v^l) := IT_{v^l}$. Note that $\forall l$, $IT_l \subset IT_{v_l}$ (in IT_{v_l} we have more targets, the ones obtained by subsequent verification queries).

Moreover, clearly $IT = \bigcup_{l=1}^{I} IT_{v_I}$ and $\forall l, l' = 1, ..., I$ $IT_{v_l} \cap IT_{v_{l'}} = \emptyset$ iff $l \neq l'$.

Thus, we can bound $\sum_{j=0}^{q_V} |IT_j| \leq \sum_{j=0}^{q_V} |IT_{v_j}|$.

Let $IW(v^i) := \{m$ used in verification queries s.t. $\mathsf{H}_s(m) = u \| v^i$ for any $u\}$.

Thus, $\sum_{j=0}^{q_V} |IT_{v^j}| = \sum_{j=0}^{I} |IW(v_j)||IT_{v_j}|$.

Clearly $\sum_{I=1}^{q_V} |IW(v^i)| = N - 1$. Moreover, $\forall i = 1, ..., I$ $|IT_{v_i}| \leq N$. In fact, we are in Game 3, where event $N\text{-}HHC$ has not happened, thus, we may have at most N different target y^i for each v^i. Thus,

$$\sum_{j=0}^{I} |IW(v_j)| \cdot |IT_{v_j}| \leq N \sum_{j=0}^{I} |IW_{v_j}| = N|T| = Nq_V.$$

Computing $|\Pr[E_3] - \Pr[E_4]|$**.** Putting the previous 2 paragraphs together, we obtain that

$$|\Pr[E_3] - \Pr[E_4]| \leq \epsilon_{\mathsf{sUP\text{-}L2}} \sum_{j=0}^{q_V} |IT_j| \leq \epsilon_{\mathsf{sUP\text{-}L2}} l q_V.$$

Computing $|\Pr[E_3] - \Pr[E_4]|$**.** Putting the previous 2 paragraphs together, we obtain that

$$|\Pr[E_3] - \Pr[E_4]| \leq \epsilon_{\mathsf{sUP\text{-}L2}} \sum_{j=0}^{q_V} |IT_j| \leq \mu q_V \cdot \epsilon_{\mathsf{sUP\text{-}L2}}.$$

Game 5 Game 5 is Game 5, except that we abort if there is a fresh and valid verification query.

Since a forgery cannot come from a previous verification query using a y or a $\tilde{y}$ already defined due to what we did in Game 3 and 4, we have that, the only possibility is that either $\tilde{y}^j = y^j$ for a verification query when there have never been a query to $\mathsf{LEval}(v^j, y^j)$ or one to LInv on input (v^j, τ^*), for any τ^*, resulting in y^j. More formally, we want to consider the event GT

$$GT := \left\{\exists i \in \{1, ..., q_{V+1}\} \text{ s.t. } y^i = \tilde{y} \text{ and } (m^i, \tau^i) \text{ is fresh}\right\}$$

and we abort if event GT happens. Let E_5 be the event that the adversary wins the game; that is, the output of the game is 1.

Transition between Game 4 and Game 5. We build a sequence of $q_V + 2$ games: Game 3^0,...,Game 3^{q_V+1}.

Game 4^j. Game 4^j is Game 4, where we abort if one of the first j verification queries has triggered event GT, that is fresh and valid. Thus, Game 4^0 is Game 4, while Game 4^{q_V+1} is Game 5. We denote with GT_j the fact that after j verification queries event GT has been triggered.
Let E_4^j be the event that the adversary wins this game.

Transition between Game 4^j and Game 4^{j+1}. Clearly, Game 4^j and Game 4^{j+1} are identical if the $j+1$-th verification query does not trigger event GT. Clearly, $\left|\Pr[E_4^j] - \Pr[E_4^{j+1}]\right| \leq \Pr[GT_{j+1}|\neg GT_j]$. To bound the probability that event $GT_{j+1}|\neg GT_j$ happens we build a $(q_{\mathsf{LF}}, q_{\mathsf{Eval}}, q_{\mathsf{Inv}}, t_2)$-$\mathsf{sUP\text{-}L2}$-adversary EE^j against F.

The $(q_{\mathsf{LF}}, q_{\mathsf{Eval}}, q_{\mathsf{Inv}}, t_2)$-$\mathsf{sUP\text{-}L2}$-adversary EE^j. EE^i has to find a valid triple (x, tw, y) for F_{k_2} which is fresh and valid.
At the start of the game, she picks uniformly at random a key $s \xleftarrow{\$} \mathcal{HK}$ for the hash function, which she relays to A^{L}. In addition, she picks a key k_1. Moreover, she has a list $\mathcal{S}$, which is empty at the start of the game.
When A^{L} does a query to L on input $(tw, x; k')$ either to Eval or to Inv, FF^j does the same query. She receives the leakage L, which she forwards to A. [23]
When A does a tag-generation query on input m^i, EE^j (1) simply computes $u^i \| v^i = h^i = \mathsf{H}_s(m^i)$, then, (2) she computes $y^i = \mathsf{F}_{k_1}^{v^i}(u^i)$and collects the leakage $\mathsf{l}_{\mathsf{e}}^{i,0} = \mathsf{L}_{\mathsf{Eval}}(v^i, u^i; k_1)$, after that (3), she queries his oracle on input (v^i, y^i) receiving τ^i and collects the leakage $\mathsf{l}_{\mathsf{e}}^{i,1}$, finally (4), she answers A τ^i and the leakage $L_M^i = (y^i, \mathsf{l}_{\mathsf{e}}^{i,0}, \mathsf{l}_{\mathsf{e}}^{i,1})$; moreover, she adds $\{(m^i, \tau^i)\}$ to $\mathcal{S}$.
This takes time $t_{\mathsf{H}} + t_{\mathsf{F}} + t_{\mathsf{L(F)}}$ and she does one query to LEval.
When A asks one of the first j verification query on input (m^i, τ^i), EE^j first, (1) computes $u^i \| v^i = h^i = \mathsf{H}_s(m^i)$, then, (2) she computes $y^i = \mathsf{F}_{k_1}^{v^i}(u^i)$and collects the leakage $\mathsf{l}_{\mathsf{e}}^{i,0} = \mathsf{L}_{\mathsf{Eval}}(v^i, u^i; k_1)$, after that (3), she queries his oracle LInv on input (v^i, τ^i) receiving $\tilde{y}^i$ and collects the leakage $\mathsf{l}_{\mathsf{i}}^{i,1}$. Finally (4), if $y^i = \tilde{y}^i$ and $\{(m^i, \tau^i)\} \in \mathcal{S}$ (4a), EE^j answers $\top$ and the leakage $L_V^i = (y^i, \tilde{y}^i, \mathsf{l}_{\mathsf{e}}^{i,0}, \mathsf{l}_{\mathsf{i}}^{i,1})$ to A; if $y^i = \tilde{y}^i$ and $\{(m^i, \tau^i)\} \notin \mathcal{S}$ (4b), EE^j aborts; otherwise (4c), $\perp$. Moreover, EE^j gives the leakage $L_V^i = (y^i, \tilde{y}^i, \mathsf{l}_{\mathsf{e}}^{i,0}, \mathsf{l}_{\mathsf{i}}^{i,1})$ to A.
This takes time $t_{\mathsf{H}} + t_{\mathsf{F}} + t_{\mathsf{L(F)}}$. Moreover, FF^j does one query to LInv.
When A asks his $j+1$-th verification query on input $(m^{q_V+1}, \tau^{q_V+1})$, EE^j computes (1) $u^{j+1} \| v^{j+1} = h^{j+1} = \mathsf{H}_s(m^{j+1})$, then (2), then, (2) she computes $y^{j+1} = \mathsf{F}_{k_1}^{v^{j+1}}(u^{j+1})$ and collects the leakage $\mathsf{l}_{\mathsf{e}}^{i,0} = \mathsf{L}_{\mathsf{Eval}}(v^{j+1}, u^{j+1}; k_1)$,. Finally, (3) she outputs $(y^{j+1}, v^{j+1} \| 1, \tau^{j+1})$ as her prediction. This takes time t_{H}.

[23] For simplicity, we suppose that A tries only to model the leakage of F and not the leakage of the full MAC. In fact, she can compute everything. Anyway the extension is straightforward.

Thus, in total, EE^i does at most q_{L} query to L, $q_M \leq q_M$ to LEval and $j \leq q_V$ to LInv; moreover, she needs time at most $t+(q_M+j+1)t_{\mathsf{H}}+(q_V+q_M+1)(t_{\mathsf{F}}+t_{\mathsf{L}(\mathsf{F})} \leq t_2$.

Bounding $|\Pr[E_4^j] - \Pr[E_4^{j+1}]|$**.** Note that EE^i simulates correctly Game 4^j for A^{L}.
We observe that if event $GT_{j+1}| \neq GT_j$ happens, the adversary EE^j does a correct prediction for the TBC F. F is a $(q_{\mathsf{L}}, q_{\mathsf{Eval}}, q_{\mathsf{Inv}}, t_2, \epsilon_{\mathsf{sUP\text{-}L2}})$-strongly unpredictable with leakage, we can bound:

$$|\Pr[E_4^j] - \Pr[E_4^{j+1}]| \leq \Pr[GT^{j+1}| \neq GT_j] \leq \epsilon_{\mathsf{sUP\text{-}L2}}.$$

Bounding $|\Pr[E_4] - \Pr[E_5]|$**.** Iterating, we obtain

$$|\Pr[E_4] - \Pr[E_5]| \leq \sum_{j=0}^{q_V+1} |\Pr[E_4^j] - \Pr[E_5^{j+1}]| \leq \sum_{j=0}^{q_V+1} \epsilon_{\mathsf{sUP\text{-}L2}} = (q_V+1)\epsilon_{\mathsf{sUP\text{-}L2}}.$$

Concluding the proof. Putting everything together, since $\Pr[E_5] = 0$ (we abort if the adversary is winning Game 5), we conclude the proof since

$$\Pr[E_0] = \sum_{i=0}^{4} |\Pr[E_i] - \Pr[E_{i+1}]| + |\Pr[E_5]| =$$
$$\epsilon_{\mathsf{CR}(n)} + \epsilon_{\mu\text{-}\mathsf{CR}(n)} + \epsilon_{\mathsf{sUP\text{-}L2}}(\mu+1)q_V + \epsilon_{\mathsf{sUP\text{-}L2}}\mu q_V + (q_V+1)\epsilon_{\mathsf{sUP\text{-}L2}} + 0 =$$
$$\epsilon_{\mathsf{CR}(2n)} + \epsilon_{\mu\text{-}\mathsf{CR}(n)} + [(2\mu+1)q_V + \mu + 1]\epsilon_{\mathsf{sUP\text{-}L2}}.$$