Automated Analysis and Synthesis of Message Authentication Codes

Stefan Milius
Friedrich-Alexander-Universität Erlangen-Nürnberg
stefan.milius@fau.de

Dominik Paulus
Friedrich-Alexander-Universität Erlangen-Nürnberg
dominik.paulus@fau.de

Dominique Schröder
TU Wien
dominique.schroeder@tuwien.ac.at

Lutz Schröder
Friedrich-Alexander-Universität Erlangen-Nürnberg
lutz.schroeder@fau.de

Julian Thomas*
Friedrich-Alexander-Universität Erlangen-Nürnberg
julian.thomas@fau.de

***Abstract*—Message Authentication Codes (MACs) represent a fundamental symmetric key primitive, serving to ensure the authenticity and integrity of transmitted data. As a building block in authenticated encryption and in numerous deployed standards, including TLS, IPsec, and SSH, MACs play a central role in practice. Due to their importance for practice, MACs have been subject to extensive research, leading to prominent schemes such as HMAC, CBCMAC, or LightMAC. Despite the existence of various MACs, there is still considerable interest in creating schemes that are more efficient, potentially parallelizable, or have specific non-cryptographic attributes, such as being patent-free.**

In this context, we introduce an automated method for analyzing and synthesizing MAC schemes. In order to achieve this goal, we have constructed a framework that restricts the class of MACs in such a way that it is sufficiently expressive to cover known constructions, yet also admits automated reasoning about the security guarantees of both known and new schemes. Our automated analysis has identified a novel category of MACs, termed "hybrid" MACs. These MACs operate by processing multiple blocks concurrently, with each block managed by a different, specified MAC scheme. A key finding is that in certain scenarios, the hybrid MAC marginally outperforms the simultaneous operation of the individual MACs. This improvement is attributed to the hybrid approach exploiting the strengths and compensating for the weaknesses of each distinct MAC scheme involved. Our implementation confirms that we have successfully identified new schemes that have comparable performance with state-of-the-art schemes and in some settings seem to be slightly more efficient.

***Index Terms*—symmetric-key authentication, program synthesis, pseudorandom function**

I. Introduction

Message Authentication Codes MAC are fundamental cryptographic building blocks that serve as the symmetric key counterpart to digital signature schemes. The generation and validation of a tag T for a given message M require the involvement of a private key K. MACs are utilized extensively in practical applications, serving both as essential components in authenticated encryption [1] and as integral parts of various established standards, including TLS [2], IPsec [3], and SSH [4]. Given their significance, MACs have been subject to extensive research, leading to the discovery of a diverse array of constructions based on various building blocks. These realizations include block cipher-based approaches, such as ECBCMAC [5], hash-based techniques, such as HMAC [6], [7], [8], and modern variants leverage special mathematical concepts, such as SpongeMAC [9]. Many of these schemes are standardized emphasizing their importance in real-world applications [8], [10], [11]. In the design of efficient MAC schemes, there is a risk of overlooking potential opportunities to enhance efficiency. For instance, CBCMAC [12] has a long history of improving both efficiency and security. CBCMAC is vulnerable to length extension attacks, while ECBCMAC [13] addresses these vulnerabilities. XCBCMAC [14] advances the design by requiring three independent keys. Subsequent developments include TMAC [15], which requires two keys, and OMAC [16], which operates with just one key.

In this work, we present an *automated* approach for analyzing and synthesizing MAC schemes. Our methodology follows a graph-based approach similar to an automated approach for authenticated encryption schemes by Hoang, Katz, and Malozemoff [17]. However, their approach relies on tweakable blockciphers, which are not used in the construction of MAC schemes and therefore cannot be used to cover current MAC schemes.

At a high level, the evaluation of the MAC is represented by a directed acyclic graph in which each node represents an instruction, such as an XOR of two values. The MAC must process messages of arbitrary length, which is modeled by splitting the MAC into three parts: initialization, iteration, and finalization. As these names suggest, initialization and finalization are responsible for pre- and post-processing, respectively, while the iteration block is a fixed description that iterates over all message blocks.

We develop a type system for the nodes of such graphs and define constraints on how nodes can be typed based on their parents' types. We then demonstrate that any well-typed graph defines a secure MAC scheme. This enables us to automatically

* Corresponding author

analyze a given scheme by checking whether the graph defining the scheme can be properly typed. Building on this, we can synthesize schemes by enumerating valid graphs and analyzing each one to determine its security.

Many existing MAC schemes satisfy our criteria, demonstrating that our framework is not overly restrictive. In particular, we obtain automatic security proofs for ECBCMAC, FCBCMAC, LightMAC, NMAC, XCBCMAC, OMAC, and TMAC. We also cover HMAC under the idealized assumption of two independent keys.

We implemented our framework in C++14 and provide the source code under an open source license[1]. This allowed us to synthesize thousands of secure MAC schemes, hundreds of which are optimal in terms of the number of calls to the underlying pseudorandom function, comparable to OMAC or XCBCMAC. We also obtain similar optimality results with respect to the number of keys. These schemes are provably secure, as verified by our analysis tool, and come with concrete security bounds.

Our automated synthesis has revealed a new class of MACs, which we term *hybrid* MACs. A MAC is classified as hybrid if the iteration graph processes multiple message blocks, which opens up the possibility to combine different MAC schemes. Notably, the parallel hybrid MACs stand out for their performance. Our findings indicate that these hybrid MACs can slightly outperform individual MACs in certain setups. This improvement arises because the hybrid approach leverages the strengths and mitigates the weaknesses of each individual scheme.

Among the many possible parallel hybrid MACs, this paper takes a closer look at the combination of CBCMAC variants and LightMAC. We implement two of these hybrid schemes and find that their running times are comparable to those of LightMAC and PMAC [18]. In some settings, particularly with larger messages, they even slightly outperform them. Therefore, these schemes may appeal to practitioners seeking efficient, straightforward, and patent-free MAC schemes for enhanced efficiency optimization in large-scale computational facilities.

A. Related Work

Barthe et al. [19] study the automated analysis and Tiwari et al. [20] the automated synthesis of cryptographic primitives, both with application to RSA-based encryption schemes. Following that, Barthe et al. consider an automated analysis of assumptions in generic groups [21], which is foundational for their subsequent work on the automated synthesis of signatures with certain properties [22]. Regarding signatures, Akinyele et al. [23] present an automated tool for generating batch verification schemes. Gagné et al. [24] discuss an approach based on Hoare Logic to verify the almost universality property of a hash function automatically.

Carmer and Rosulek [25] provide a more general approach to automated analysis and synthesis within the Linicrypt framework, by describing algorithms with linear operations and random oracle access algebraically. Due to the random oracle, the analysis of hash functions by McQuoid et al. [26] and Javar and Kapron [27] is evident within the Linicrypt framework. Similarly, Hollenberg et al. [28] analyses the instantiation of the random oracle with block ciphers.

Another common approach, which we expect to provide more modularity and detailed analysis, is the graph-based description of algorithms. Malozemoff, Katz, and Green [29] pioneered this research field for cryptography by automatically generating secure block-cipher modes. Later, Hoang, Katz, and Malozemoff [30] extended this approach to authenticated encryption schemes. While the Encrypt-then-Mac paradigm [1] can provide authenticated encryption, Hoang, Katz, and Malozemoff [30] focus on efficient authenticated encryption schemes that do not separate encryption and authentication. Although one could devise a function that maps from the generated authenticated encryption schemes to MACs, due to the focus on efficiency and correctness in [30], this would only cover a small subset of the possible MAC schemes. Summing up, a comprehensive coverage of MAC constructions is missing.

II. Preliminaries

Notation. Let $\mathsf{maps}(X, Y)$ be the set of all functions from domain X to codomain Y. For a finite set S, we denote sampling an element x uniformly from S by $x \leftarrow\!\$\ S$. When n is some integer less than 2^ℓ, then $\langle n \rangle_l$ is the binary representation of n when padded to exactly ℓ bits. For two bit-strings x and y, we denote concatenation with $x \| y$, and if x and y are of the same length, $x \oplus y$ denotes their bitwise XOR. We use $\mathsf{negl}(n)$ to describe an arbitrary *negligible* function $f\colon \mathbb{N} \to \mathbb{R}^+$, that is, a function that decreases faster than the inverse of any polynomial. The security parameter is λ, and when referring to something as negligible, we mean negligible in λ.

We use the code-based game-playing framework of Bellare and Rogaway [31]. Here, an adversary $\mathcal{A}$ always uses a *probabilistic polynomial time* PPT algorithm. We denote the execution of $\mathcal{A}$ with *Oracle* access to $\mathcal{O}_1, \ldots, \mathcal{O}_n$ using inputs $x_1, \ldots, x_m$, and output b by $\mathcal{A}^{\mathcal{O}_1,\ldots,\mathcal{O}_n}(x_1, \ldots, x_m) = b$.

Pseudorandom Functions and Permutations. A function $\mathsf{F}\colon \{0,1\}^\lambda \times \{0,1\}^m \to \{0,1\}^l$ is a *pseudorandom function* PRF if every PPT attacker $\mathcal{A}$ can distinguish F from a real random function only with negligible probability [32]. A random function is chosen uniformly from the same function family, i.e., the set of all functions with the same mapping. If, additionally, $m = l$ and $F(K, \cdot)$ is bijective for every $K \in \{0,1\}^\lambda$, then F is a *pseudorandom permutation* PRP [33].

In the following, we consider PRP as an abstraction of block ciphers. Regarding block ciphers, we consider variable-length messages M, denoting the ith block as $M[i]$. Unless otherwise specified, we generally use the designators F for PRF, P for PRP, f, g for arbitrary functions, and p for bijective functions (permutations).

Hash Functions. We consider a keyed *hash function* $\mathsf{H}\colon \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^l$, compressing arbitrary length

[1]During review, the code is provided with the supplementary material. After the paper's acceptance, we provide an open-source link here instead.

input to an output of fixed length l. We assume $K \leftarrow\!\$\ \{0,1\}^\lambda$ in this paragraph. Carter and Wegman [34] introduced the concept of *universality* of hash functions.

The idea is that a function family is universal if a certain property holds for any function randomly chosen from that family. For simplicity, we refer to a fixed function instantiated by a uniformly chosen key, rather than explicitly to the entire function family.

Specifically, we call a hash function H *universal* if for all $x, y \in \{0,1\}^*$ such that $x \neq y$, we have $\Pr[\mathsf{H}(K,x) = \mathsf{H}(K,y)] \leq 2^{-l}$. This can be understood as a more general definition of *collision resistance* [32]. A hash function is *strongly universal* if for any $x_1, x_2 \in \{0,1\}^*$ such that $x_1 \neq x_2$ and any $y_1, y_2 \in \{0,1\}^l$, we have $\Pr[\mathsf{H}(K,x_1) = y_1 \wedge \mathsf{H}(K,x_2) = y_2] = 2^{-2l}$ [35]; this can be weakened to *ϵ-almost (strong) universality*, where the collision probability is relaxed from $1/2^l$ to $\epsilon/2^l$ with $\epsilon \geq 1$ [36]. A computational relaxation is provided by Bellare et al. [7]: The hash function H is *computationally almost universal* if the advantage $\mathsf{Adv}^{au}_{\mathcal{A},\mathsf{H}} = \Pr\left[\mathcal{A}^{\mathsf{H}(K,\cdot)} = (x,y) \text{ where } x \neq y : \mathsf{H}(K,x) = \mathsf{H}(K,y)\right]$ is negligible.

Message Authentication Codes. A *message authentication code* MAC is an algorithm computing a tag to verify a message's integrity and authenticity. A message authentication code Π_{MAC} consists of the following three PPT algorithms:

- $\mathsf{KGen}\colon \{0,1\}^\lambda \to \{0,1\}^\lambda$**:** The probabilistic *key-generation* algorithm outputs a key K depending on λ.
- $\mathsf{Mac}\colon \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^*$**:** The *tag-generation* algorithm takes as input a key key and a message M; it returns a tag T. We call the construction *fixed-length* if the message length is fixed by some value.
- $\mathsf{Vrfy}\colon \{0,1\}^\lambda \times \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$**:** The deterministic *verification* algorithm takes as input a key K, a message M, and a tag T. It outputs a bit b, where $b = 1$ confirms the validity of T and $b = 0$ indicates invalidity.

For *correctness* we require $\mathsf{Vrfy}(K, M, \mathsf{Mac}(K,M)) = 1$ for all $\lambda \in \{0,1\}^*$, all messages $M \in \{0,1\}^*$, and every key K output by $\mathsf{KGen}(1^\lambda)$. We assume canonical verification for all deterministic MAC schemes unless otherwise noted, i.e., the verification recomputes the MAC and compares the result with the provided tag.

Regarding security, we use the notion of strong unforgeability. Intuitively, it states that forging a new and valid message-tag pair for a MAC scheme is difficult, even if the attacker has access to an oracle that computes tags on messages of its choice. We define the corresponding game $\mathsf{SUF}_{\mathcal{A},\Pi_{\mathsf{MAC}}}$ as follows:

$\mathsf{SUF}_{\mathcal{A},\Pi_{\mathsf{MAC}}}(\lambda)$	$\mathcal{O}_{\mathsf{Mac}}(K,M)$
1 : $\mathcal{Q} \leftarrow \emptyset, \quad K \leftarrow\!\$\ \mathsf{KGen}(1^\lambda)$	1 : $T \leftarrow \mathsf{Mac}(K,M)$
2 : $(M,T) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Mac}}(K,\cdot)}$	2 : $\mathcal{Q} \cup \{(M,T)\}$
3 : **if** $((M,T) \notin \mathcal{Q}) \wedge (\mathsf{Vrfy}(M,T) = 1)$	3 : **return** T
4 : **return** 1	
5 : **return** 0	

A message authentication code $\Pi_{\mathsf{MAC}} = (\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vrfy})$ is *strong existentially unforgeable under an adaptive chosen-message attack*, or just *secure*, if

$$\Pr[\mathsf{SUF}_{\mathcal{A},\Pi_{\mathsf{MAC}}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

holds for all PPT adversaries $\mathcal{A}$.

Note that a keyed PRF directly yields a secure *fixed-length* MAC scheme [32] with canonical verification, that is, $\mathsf{Vrfy}(K,M,T)$ recomputes $\mathsf{Mac}(K,M) = T'$ and returns 1 if $T = T'$ and 0 otherwise. Our framework will utilize that the eventually stronger than necessary property of pseudo-randomness fulfills the security guarantees of the $\mathsf{SUF}_{\mathcal{A},\Pi_{\mathsf{MAC}}}$-experiment.

III. The MAC Framework

We present a graph-based abstraction of MAC schemes, where nodes represent computations and their types, determined through our automatic type inference, indicate their properties. We then establish a secure construction of variable-length MAC within this framework.

A. A Graph-Based Representation of MACs

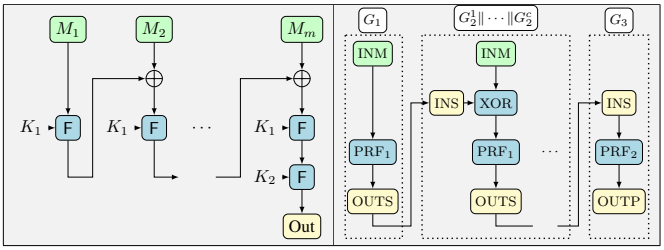

Fig. 1: The left shows a common ECBCMAC representation, while the right shows the graph-based abstraction of our framework. Dashed lines delineate the three separate subgraphs.

Our graph-based representation is inspired by common MAC constructions (We refer the reader to the appendix of the full version for a comprehensive summary of MAC constructions; see Figure 1 for a concrete example) resulting in a tripartite graph system: the *initialization* (G_1), the *iteration* (G_2), and the *finalization* (G_3). We assign each node within the subgraphs a *computation label*. These represent either calculations (e.g. concatenation CONCAT), data blocks (e.g. message blocks INM), or administrative concepts (e.g. final result OUTP). Table I shows the possible computation labels. We connect the subgraphs by an edge between the output state OUTS of G_{i-1} and the input state INS of G_i, denoted by $G_{i-1} \| G_i$. Thus, for a MAC scheme S, we create one copy of G_1 and G_3 and c copies of G_2, such that $S = G_1 \| G_2^1 \| \cdots \| G_2^c \| G_3$. The messages scheme S computes are split into blocks and assigned to the INM nodes of the subgraphs. Figure 1 shows an example of this graph-based abstraction using a common scheme.

Further Notations. We use an *evaluation function* $\mathtt{Eval}(\mathsf{F}, \mathcal{K}, G, M, i)$ to compute the result of a node with index $i \leq |V|$, where we generally write $|l|$ for the length of a list l, for a graph or subgraph G with nodes V and number of

Name	In	Out	Intuitive description
CONCAT	2	1	Concatenates both inputs
XOR	2	1	XORes both inputs
PRF_l	1	1	Invokes a PRF using key K_l
CTR	0	1	Returns a global counter value
INM	0	1	Loads an additional message block
INS	0	1	Loads previous computation state
OUTS	1	0	Stores the current computation state
OUTP	1	0	Stores the resulting computation state

TABLE I: Computation labels for the operations a node can represent, including the *In*-degree and *Out*-degree.

nodes $|V|$ (see Figure 3 for details). Here, M is a message, F instantiates the PRF_l nodes and $\mathcal{K}$ is a keyset containing the keys for the PRF_l nodes. The graph G is always completely traversed along a given order of the nodes, which is crucial because isomorphic graphs can produce distinct results due to the implicit state carried by the last CTR output. We denote the output state, the intermediate result after node i, by $\mathtt{Z}_i$. All INM-nodes that are not predecessors of node i and all INS without a corresponding OUTS node are assigned the value 0^λ. This prevents side effects, such as incrementing the counter-variable, when iterating over unused parts of the graph. As a result, the graph traversal order and nodes remain independent of the index i. Additionally, it ensures that the CTR nodes in the finalization subgraph remain unaffected by the message length. This design decision for CTR is inspired by LightMAC [37].

We denote the set of all possible computation labels by L. A *graph* $G = (V, P) \in L^* \times (\mathbb{N}^*)^*$ consists of two lists: A list V of node labels, where $V[i]$ is the label assigned to the ith node, and a list P containing a set of parent nodes for each node. Note that we assume the first index to be 1. We require $|V| = |P|$. We write $u \to v$ if $u \in P[v]$, and draw a corresponding edge in the graphical depiction, indicating that the result of node u is used as input for the computation at node v. A *path* is a list of nodes $u_1, \ldots, u_n$ such that $u_{i-1} \in P[u_i]$ for $i = 2, \ldots, n$. If a path from a node u to a node v exists, then we write $u \to^+ v$. We will use the notation $pred(u) = \{v \mid 0 < v \leq |V|, v \to^+ u\}$ for the set of all *ancestors* of a node u in a graph and $rel(u) = pred(u) \cup \{v \mid 0 < v \leq |V|, u \to^+ v\} \cup \{u\}$ for all nodes *related* to u, that is, all nodes that are either reachable from u or are predecessors of u.

Constraints. We only consider graphs that satisfy the following constraints: The graph G is acyclic. The in- and outdegrees of nodes obey the constraints in Table I. Moreover, we require the number c_i of nodes with a certain label within subgraph G_i to be as follows: for INS, $c_1 = 0$, $c_2 = 1$, $c_3 = 1$; for OUTS, $c_1 = 1$, $c_2 = 1$, $c_3 = 0$; for OUTP, $c_1 = 0$, $c_2 = 0$, $c_3 = 1$; and for INM, $c_1 \geq 0$, $c_2 \geq 1$, $c_3 \geq 0$. A graph is *well-formed* if it adheres to these constraints. A well-formed graph is necessarily an in-tree, with either OUTS or OUTP being the root and all leaves being labeled either INM, INS, or CTR. Finally, we only consider message sizes that are multiples of λ. The INM nodes represent the corresponding message blocks of size λ. We do not restrict the input size of a PRF_l node, but we assume the output to be of size λ.

B. The Type System

A *typing* for a graph G is a function T that assigns to each node i a type tuple $t = (\mathtt{TYPE}, \mathtt{DIST}, \mathtt{KEYS}, \mathtt{LEN})$ adhering to the rules of the type system as specified next. We denote the type tuple assigned to node i by t_i, and the elements of this tuple by $\mathtt{TYPE}_i, \mathtt{DIST}_i, \mathtt{KEYS}_i$, and $\mathtt{LEN}_i$, respectively. The *type* of a graph is the type tuple of its final node. In the following, we describe the four components of a type tuple in detail.

Type. The assertion about the behavior of a function computed by the subgraph rooted at node i is given by $\mathtt{TYPE}_i \in \{\bot, \mathtt{CONST}, \mathtt{AU}, \mathtt{PU}, \mathtt{PRF}\}$. Fix some graph G and a node i, then $\mathtt{TYPE}_i$ asserts the following statements:

$\mathtt{TYPE}_i$	Description
$\bot$	Universally valid
CONST	Node i has a constant value independent of the input
AU	$\mathtt{Eval}(\mathsf{F}, \mathcal{K}, G, M, i)$ is an universal function
PU	$\mathtt{Eval}(\mathsf{F}, \mathcal{K}, G, M, i)$ is strongly universal function
PRF	$\mathtt{Eval}(\mathsf{F}, \mathcal{K}, G, M, i)$ is a secure PRF

Section II describes the properties AU (universal function), PU (strongly universal function), and PRF (pseudorandom function) in more detail. We introduce CONST to differentiate between INM and CTR nodes in the respective subgraphs since an adversary may control the input message but not the counters.

Dist. In contrast, the distribution property $\mathtt{DIST}_i \in \{\bot, \mathtt{CR}, \mathtt{AIU}\}$ relates two different nodes in a graph G, which is especially relevant for nodes with two parents (XOR and CONCAT). Thus, $\mathtt{DIST}_i$ asserts the following statements:

$\mathtt{DIST}_i$	Description
$\bot$	Universally valid
CR	Node i is collision-free for all unrelated nodes
AIU	Node i is almost independent and uniform

We regard the assignment $\mathtt{DIST}_i = \mathtt{CR}$ as correct if the probability for a collision between the value at node i and all unrelated nodes typed CR or AIU is negligible. We specify the requirements for $\mathtt{DIST}_i = \mathtt{AIU}$ as follows: First, the values at node i have to be uniformly distributed. Second, all values at pairwise unrelated nodes that are also cast $\mathtt{DIST} = \mathtt{AIU}$ have to be jointly independently distributed. In both cases, the deviation is at most negligible.

Intuitively, TYPE is an assertion about how the value at some node relates to values at the same node for different input messages. In contrast, DIST relates the values at some node with those at other nodes for the same and different input messages.

The contrast between TYPE and DIST may appear counterintuitive. This becomes more evident by considering the example graphs in Figure 2. In the left subgraph, both results of the PRF invocations (edge (1) and (2)) are the result of a PRF invocation on the input for the corresponding subgraph. This would justify $\mathtt{TYPE}_1 = \mathtt{TYPE}_2 = \mathtt{PRF}$ for both nodes. However, they are not distributed independently. An adversary could input the same message block for both INM nodes. Then, the

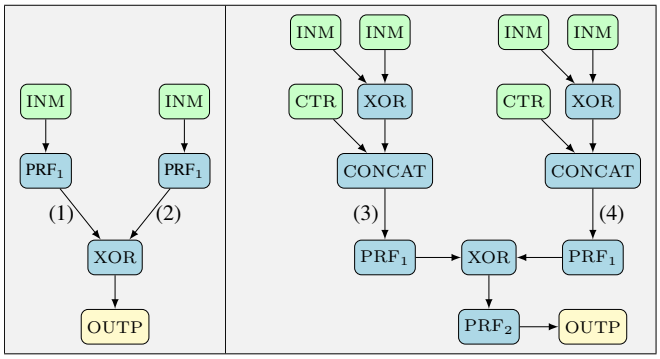

Fig. 2: Example graphs to emphazise the distinction between TYPE and DIST.

PRF nodes output the same value, and the output of XOR consists of only zeroes.

In the right graph, both inputs to CONCAT are collision-resistant, as the CTR node indicates. This justifies $\texttt{DIST} = \texttt{AIU}$ for the result of the output nodes of edges (3) and (4). However, the functions computed by these nodes fall short of achieving AU. Assume input pairs like (A, A) and (B, B). Again, the XOR block output only consists of zeros. Thus, the appropriate typing for edges (3) and (4) is $\texttt{TYPE}_3 = \texttt{TYPE}_4 = \bot$.

The labels TYPE and DIST may be refined stepwise. Specifically, in Lemma III.1 we identify a (partial) ordering on TYPE and DIST such that $\bot < \texttt{CONST}$, $\bot < \texttt{AU} < \texttt{PU} < \texttt{PRF}$, and $\bot < \texttt{CR} < \texttt{AIU}$, respectively.

Keys and Length. $\texttt{KEYS}_i \subseteq \mathbb{N}$ denotes the set of keys used in the computation of $\texttt{Eval}(\mathsf{F}, \mathcal{K}, G, M, i)$. If $K_l \notin \texttt{KEYS}_i$, then there is no path from a node labeled PRF_l to node i. Finally, $\texttt{LEN}_i \in \mathbb{N}$ denotes the length of the corresponding output block at node i, thus $\texttt{LEN}_i$ is the length of the block output by $\texttt{Eval}(\mathsf{F}, \mathcal{K}, G, M, i)$.

C. Type Inference on Graphs

We now provide an intuition for the type inference algorithm. We use this algorithm to assign the types from the type system to the nodes of a graph, in order to automatically analyze whether a graph represents a secure MAC scheme. See Appendix VII for a comprehensive summary of technical details.

First, we summarize the components of our framework and their relations. We are given an acyclic graph, which represents a sequence of calculations based on the computation labels of the nodes. The *evaluation function* (see Figure 3) performs the calculations any subgraph of the given graph represents. The *type inference* analyzes the properties of this function. In doing so, it assigns the types of the type system to the nodes. If we analyze a graph resembling a MAC scheme completely, and $\texttt{TYPE} = \mathrm{PRF}$ holds for the last node, then we can conclude security for this scheme, which we prove in Theorem III.8.

The algorithm begins with the INM and CTR nodes, which have no parent nodes. Initially, we assign them the fundamental properties that apply to message blocks and counters. For INM this includes $\texttt{TYPE} = \texttt{AU}$. Considering the evaluation function in Figure 3 we see that on input M (of size λ), the function assigns M to the intermediate result and outputs it afterward. This is the functionality of the identity function, which is (almost computationally) universal AU since two different inputs will always result in two different outputs.

The proof of Theorem III.3 summarizes these arguments for the (nontrivial) assignments and provides an argument for the soundness of the inference algorithm. Since our given graph is acyclic, we can proceed by well-founded induction over it. In the base case we treat the INM and CTR notes. In the induction step we assume that the parent nodes of a node have already been assigned a typing, and we prove that the typing of that node is correctly inferred by the algorithm.

The functionality of CONCAT and XOR nodes is combining two subgraphs. As these strongly depend on the parent nodes, the associated assignment rules are detailed. Note that CONCAT takes over the weaker of the two properties, whereas XOR can adopt the stronger property.

We base the security of our construction on pseudorandomness. Consequently, we need PRF nodes to provide these properties and 'strengthen' (see Lemma III.1) the types, if possible. Besides some trivial cases, we rely on a result by Mihir Bellare [7]. To be more specific, if the evaluation of the subgraph rooted in the parent node is a computational almost universal function (i.e. $\texttt{TYPE} = \texttt{AU}$ holds for the parent node), and the PRF node has an independent key, then we can assign $\texttt{TYPE} = \mathrm{PRF}$ to this node. In the case of key dependency, we conclude $\texttt{TYPE} = \texttt{PU}$.

We use the administrative nodes OUTS, INS, and OUTP to connect the subgraphs and indicate the final result. Hence, their type inference is straightforward. Besides some correctness checks, those nodes adapt the typing of their parent node.

D. Proof of Correct Typing

This section provides the formal proof of the soundness of the type inference algorithm. We start by formally stating and proving the (partial) ordering of TYPE and DIST mentioned in Section III-B. We say t_1 is valid (i.e. correct)for a node in a given (sub)graph if the type inference assigns the property of t_1. We say t_1 is stronger than t_2 and t_2 is weaker than t_1, if $t_2 < t_1$ holds.

Lemma III.1 (Transitive implication order)**.** *Let $<$ be a partial order on* TYPE *and* DIST *such that $\bot < \texttt{CONST}$, $\bot < \texttt{AU} < \texttt{PU} < \texttt{PRF}$, and $\bot < \texttt{CR} < \texttt{AIU}$, respectively. If t_1 is a valid* TYPE *or* DIST *for node i, and $t_2 < t_1$, then the assertion of t_2 is also correct for i.*

Proof. For $\bot < \texttt{CR}$, $\bot < \texttt{CONST}$, and $\bot < \texttt{AU}$, this is trivial. For $\texttt{CR} < \texttt{AIU}$, the assertion for nodes to be typed CR also asserts the collision resistance with AIU typed nodes. The probability of two unrelated AIU typed nodes colliding is $2^{-\lambda}$.

The relation $\texttt{AU} < \texttt{PU}$ as described in Section II follows from the definition [35]. Regarding $\texttt{PU} < \texttt{PRF}$, a PRF must fulfill strong universality; otherwise, it could be distinguished. We simplify the reasoning of $\texttt{PU} \not> \texttt{PRF}$ within the framework,

using the necessity of key independence for a node to be typed PRF, which is not the case for PU. □

For the main proofs, we use the function IB, which outputs the number of message blocks as input for a (sub-)graph and is defined as follows:

Definition III.2 (Number of In-blocks)**.** Consider a graph G and a message M. Let $M_G = [k \mid V(k) = \text{INM}]$ be the list of INM-nodes contained in G. Then, $\texttt{IB}(G, M, i)$ is the list of input message blocks consumed when evaluating the subgraph rooted at i; in symbols: $[M_x \mid x \in [1; |M_G|] \wedge M_G[x] \rightarrow^+ i]$ if $i < |V|$ or with $\texttt{IB}(G, M, |V| - 1)$ otherwise.

The idea of the type interference is that we can automatically evaluate whether any given graph implements a secure MAC. For this, the typing of an arbitrary graph must be correct and indicate, that the MAC scheme fulfills pseudorandomness (node OUTP has TYPE = PRF). We base our proof of correct typing on two theorems. Theorem III.3 states the correctness of the typing rules, which we will prove using induction on arbitrary graphs and under the assumption of a real random function instead of a PRF.

Lemma III.4 ensures that we can indeed replace the PRF with a real random function. We prove this using a game-hopping argument. Figure 9 in Appendix VII summarizes the corresponding games. For a graph G and randomly sampled keys K, we use Game_3, which is presented in Figure 3, as an oracle for computing the graph with $\texttt{Eval}(\mathsf{F}, \mathcal{K}, G, \cdot, \cdot)$.

Theorem III.3 (Correctness of type inference)**.** *For any well-formed (see Constraints in Section III-A) graph G with a typing assigned by the type inference from Section III-C, the evaluation function as implemented in Game_3 satisfies the assertions implied by the types of* TYPE*,* DIST*,* KEYS*, and* LEN*.*

Proof. We perform an induction proof on the number of remaining nodes in an arbitrary graph. The base case consists of the nodes without parents, which are INM and CTR. All other nodes are part of the induction step since at least one parent exists. Therefore, we assume the parent node(s) have been correctly typed. This is possible because the type inference traverses the circle-free graph in a certain topological order. LEN and KEYS are correct by construction and are not further discussed. The administrative nodes, INS, OUTS, and OUTP, are not considered, as they do not change the types and are implicitly correct based on the induction hypothesis of correctly typed parents.

We structure the proof modularly by making a case distinction for the different node labels. To emphasize this, we slightly simplify the notation from a common induction proof: We use i to denote any node in any graph and j and k to denote its parents if they exist. For every case, we analyze TYPE and then DIST.

$V(i) = \text{INM}$: Assigning $\texttt{TYPE}_i = \texttt{AU}$ is correct because a single INM-node represents the identity function, which is universal. Assigning $\texttt{DIST} = \bot$ is generally correct. The argument is the same for the base case and induction step.

$\mathsf{Game}_3(V, P, \lambda)$

for $k \in 1, \ldots, |\mathcal{K}| : f_k \leftarrow\!\!\$\ \mathsf{maps}(\{0,1\}^*, \{0,1\}^\lambda)$
$s = 1$

On query (M, i)

$incnt,\ cnt,\ state,\ PM_s = 1,\ 1,\ 0^\lambda,\ M$
for $j \in \{1, \ldots, |V|\}$:
  if $V[j] = \text{INM}$:
    if $j \notin pred(i) : \mathtt{z}_{sj} = 0^\lambda$
    else $: \mathtt{z}_{sj} = M[incnt],\ incnt = incnt + 1$
  elseif $V[j] = \text{CTR}$:
    if $\exists x > j, V[x] = \text{OUTS} : \mathtt{z}_{sj} = \langle cnt \rangle_\lambda$
    else $: \mathtt{z}_{sj} = \langle 2^\lambda - cnt - 1 \rangle_\lambda$
    $cnt = cnt + 1$
  elseif $V[j] = \text{XOR} :\ (p, q) = P[j]; \mathtt{z}_{sj} = \mathtt{z}_{sp} \oplus \mathtt{z}_{sq}$
  elseif $V[j] = \text{PRF}_l$:
    $(p) = P[j]$
    if $(Error) : \mathtt{z}_{sj} \leftarrow\!\!\$\ \{0,1\}^\lambda$
    else $: \mathtt{z}_{sj} = f_l(\mathtt{z}_p)$
  elseif $V[j] = \text{CONCAT} :\ (p, q) = P[j];\ \mathtt{z}_{sj} = \mathtt{z}_{sp} \| \mathtt{z}_{sq}$
  elseif $V[j] = \text{OUTP} :\ (p) = P[j];\ \mathtt{z}_{sj} = \mathtt{z}_{sp}$
  elseif $V[j] = \text{OUTS}$:
    $(p) = P[j];\ \mathtt{z}_{sj} = \mathtt{z}_{sp};\ state = \mathtt{z}_{sj}$
    if $\neg\exists x > j, V[x] = \text{OUTS} : cnt = 1$
  elseif $V[j] = \text{INS} : \mathtt{z}_{sj} = state$
$s = s + 1$
return $\mathtt{z}_{(s-1)i}$

Fig. 3: This game represents the evaluation function Eval for Theorem III.3. It outputs the computational result at node i for message M of graph $G = (V, P)$. Here, a real random function implements the pseudorandom function. This arises from the game-hopping argument in Lemma III.4. See Figure 9 for technical details regarding an *Error* to occur.

$V(i) = \text{CTR}$: Assigning $\texttt{TYPE}_i = \texttt{CONST}$ is correct. First, CTR nodes are independent of other nodes, as they do not have a parent. Second, evaluating the same graph always yields the same counter value at index i, independently of any input to the graph.

For DIST, we show that $V(i)$ is collision-resistant to all other nodes typed CR or AIU. Considering the type inference, CR is only possible for CTR or CONCAT nodes. The construction implies the different length of CONCAT, so there can only be a collision with other CTR nodes. We assume counter nodes always output different values for different indices. A collision can only occur if there are more CTR nodes than possible values. Since the number of values depends on λ, an adversary would have to queue exponentially long messages, which contradicts the PPT assumption. Considering $\texttt{DIST} = \texttt{AIU}$, note that CTR values are constant concerning different keys and that AIU guarantees uniform distribution. Thus, the

counter's probability of colliding is $\leq 2^{-\lambda}$ for any unrelated $\texttt{AIU}$-labeled node. The arguments are the same for the base case and induction step.

$V(i) = \text{CONCAT}$: The correctness of $\texttt{TYPE}_i = \bot$ or $\texttt{TYPE}_i = \texttt{CONST}$ is implicit. If $\texttt{TYPE}_i = \texttt{AU}$, then at least one parent has $\texttt{TYPE} \geq \texttt{AU}$. We assume w.l.o.g. that this is the case for $\texttt{TYPE}_j$ and only consider $\texttt{TYPE}_j = \texttt{AU}$, which is possible due to the (partial) ordering Lemma III.1 describes. Let $e_i = \texttt{Eval}(\mathsf{F}, \mathcal{K}, G, \cdot, i)$ be the evaluation of node i for arbitrary messages and let e_j and e_k be analogous for the parent nodes. Let $M_{11} \| M_{12}$ and $M_{21} \| M_{22}$ be two messages. Then, the probability of a collision, which would contradict $\texttt{TYPE}_i = \texttt{AU}$, is

$$\begin{aligned} & e_i(M_{11} \| M_{12}) = e_i(M_{21} \| M_{22}) \\ & \Leftrightarrow e_j(M_{11}) \| e_k(M_{12}) = e_j(M_{21}) \| e_k(M_{22}) \\ & \Leftrightarrow e_j(M_{11}) = e_j(M_{21}) \wedge e_k(M_{12}) = e_k(M_{22}) \\ \Rightarrow & \Pr[e_i(M_{11} \| M_{12}) = e_i(M_{21} \| M_{22})] \\ & \leq \Pr[e_j(M_{11}) = e_j(M_{21})] \leq 2^{-\lambda}, \end{aligned}$$

where we utilize w.l.o.g. a fixed output length for e_j. Note that there is no collision for values of different sizes.

Again, $\texttt{DIST}_i = \bot$ is implicit. The other case is $\texttt{DIST}_i = \texttt{CR}$. A collision can only occur with other CONCAT nodes due to the length of the nodes. If there is a collision, parents j and k would also have a collision, which would contradict the assumption.

$V(i) = \text{XOR}$: Again, let $e_i = \texttt{Eval}(\mathsf{F}, \mathcal{K}, G, \cdot, i)$ be the evaluation of node i for arbitrary messages and let e_j and e_k be analogous for the parent nodes. We have three cases:

1) Either $\texttt{TYPE}_j = \texttt{CONST}$ or $\texttt{TYPE}_k = \texttt{CONST}$. W.l.o.g., we assume that this applies to node j. Then $\texttt{TYPE}_i = \texttt{TYPE}_k$, and $\texttt{TYPE}_k$ is either $\texttt{CONST}$, $\texttt{AU}$, $\texttt{PU}$ or $\texttt{PRF}$. All of these properties are preserved by XORing with a constant.
2) If $\texttt{TYPE}_i = \texttt{PU}$, then $\texttt{TYPE}_j \geq \texttt{PU} \wedge \texttt{TYPE}_k \geq \texttt{PU}$, and either $\texttt{KEYS}_j \cap \texttt{KEYS}_k = \emptyset$ or $\texttt{DIST}_j = \texttt{DIST}_k = \texttt{AIU}$. We first consider the latter case. Let M_1 and M_2 be different messages as input for node i with parent messages M_{11}, M_{12} for M_1 and M_{21}, M_{22} for M_2

$$\begin{aligned} & \Pr[e_i(M_1) = y_1 \wedge e_i(M_2) = y_2] \\ & = \Pr[e_j(M_{11}) \oplus e_k(M_{12}) = y_1] \\ & \quad \cdot \Pr[e_j(M_{21}) \oplus e_k(M_{22}) = y_2] \\ & = \sum_{a,b \in \{0,1\}^\lambda} \Pr\left[\begin{array}{l} e_i(M_{11}) = a \wedge e_i(M_{21}) = b \\ \wedge e_j(M_{12}) = a \oplus y_1 \\ \wedge e_j(M_{22}) = b \oplus y_2 \end{array}\right] \\ & = \sum_{a,b \in \{0,1\}^\lambda} \frac{1}{2^{2\lambda}} \cdot \frac{1}{2^{2\lambda}} = 2^{2\lambda} \cdot \frac{1}{2^{2\lambda}} \cdot \frac{1}{2^{2\lambda}} = \frac{1}{2^{2\lambda}}. \end{aligned}$$

Thus, the computation of two arbitrary results, y_1 and y_2, is pairwise independently and uniformly distributed. We use the fact that any block labeled $\texttt{DIST} = \texttt{AIU}$ is of length λ and the assertions that $\texttt{DIST} = \texttt{AIU}$ imply. The argument for $\texttt{KEYS}_j \cap \texttt{KEYS}_k = \emptyset$ is the same, except for the independence of distributions at nodes j and k, which emerges from the independent sampling of $\texttt{KEYS}_j$ and $\texttt{KEYS}_k$ and the fact that $\texttt{TYPE}_j \geq \texttt{PU}$ and $\texttt{TYPE}_k \geq \texttt{PU}$, which implies, that the paths to j and k include a PRF nodes, which are independent between both paths since $\texttt{KEYS}_j \cap \texttt{KEYS}_k = \emptyset$ holds.

3) Given $\texttt{TYPE}_i = \texttt{AU}$, we know that either $\texttt{TYPE}_j$ or $\texttt{TYPE}_k$ is $\texttt{PU}$. W.l.o.g., we assume $\texttt{TYPE}_i = \texttt{PU}$. As XOR is commutative $e_i(M) = z$ can be rewritten as $e_j(M_1) = z \oplus e_k(M_2)$. Thus, a collision happens if

$$\begin{aligned} & e_j(M_{11}) \oplus e_k(M_{12}) = e_j(M_{21}) \oplus e_k(M_{22}) \\ \Leftrightarrow & e_j(M_{11}) \oplus e_j(M_{21}) = e_k(M_{12}) \oplus e_k(M_{22}). \end{aligned}$$

Due to the parent's pairwise independent distributions $\texttt{PU}$, the probability for this collision is $2^{-\lambda}$.

$\texttt{DIST}_i = \bot$ is always correct. If $\texttt{DIST}_i = \texttt{AIU}$, parents j and k are also labeled $\texttt{AIU}$. Note that an unrelated node to i is also unrelated to j and k. Any set D in the definition of $\texttt{DIST}$ containing node i does not contain j or k, as F must consist of pairwise unrelated nodes only. Thus, the uniformity and independence of the parent nodes imply uniform distribution. In the case that w.l.o.g. only j is labeled with $\texttt{AIU}$, we know again that a PRF node exists on the path to j, that is key independent to all nodes from path k, which implies the uniform distribution.

Joint independence is achieved because if D satisfies the mentioned pairwise independence property and contains i, then D satisfies the $\texttt{DIST} = \texttt{AIU}$-assertion iff $(\mathsf{F} \backslash \{i\}) \cup \{j, k\}$. This is true due to the assumption of correctly typed parents.

$V(i) = \text{PRF}_l$: If $\texttt{TYPE}_j = \bot$ we assign $\texttt{TYPE}_i = \bot$, which is always valid per definition. If $\texttt{TYPE}_j = \texttt{CONST}$, we assign $\texttt{TYPE}_i = \texttt{CONST}$, which is valid, since the computation of PRF_l is deterministic. Besides the trivial cases, two cases arise, if $\texttt{TYPE}_j \geq \texttt{AU}$ holds:

1) For $\texttt{TYPE}_i = \texttt{PRF}$, we also need $K_l \cap \texttt{KEYS}_j = \emptyset$. By Lemma III.1, we know that $\texttt{TYPE}_j = \texttt{AU}$ would also be a valid labeling on j. Then we have $\texttt{Eval}(\mathsf{F}, \mathcal{K}, G, M, i) = \mathsf{F}(K_l, \texttt{Eval}(\mathsf{F}, \mathcal{K}, G, M, j))$. By applying Lemma 3.2 from Mihir Bellare [7] (see Appendix VIII), $\texttt{Eval}(\mathsf{F}, \mathcal{K}, G, M, i)$ implements a PRF and consequently $\texttt{TYPE}_i = \texttt{PRF}$ is correct.
2) For $\texttt{TYPE}_i = \texttt{PU}$, we show forall $x \in \{0,1\}^\lambda$, $y \in \{0,1\}^\lambda$, and arbitrary messages M_1, M_2 that $\Pr[\texttt{Eval}(\mathsf{F}, \mathcal{K}, G, M_1, i) = x] \cdot \Pr[\texttt{Eval}(\mathsf{F}, \mathcal{K}, G, M_2, i) = y] = 2^{-2\lambda}$ holds.

We have two cases for the evaluation state $\mathtt{Z}_{si}$. Recap that, for q message queries, $\mathtt{Z}_{ab}$ is the value of node b by $\texttt{Eval}$ when queried for query a and that we use $\texttt{Game}_3$ and the corresponding notations for the evaluation of Eval: 1. $\mathtt{Z}_{si}$ is uniformly and independently sampled from $\{0,1\}^\lambda$. 2. Otherwise $\neg \exists w, (\mathtt{Z}_{wp} = \mathtt{Z}_{sp} \wedge \texttt{IB}(G, M, i) \neq \texttt{IB}(G, PM_w, i))$ holds. This means that $\texttt{Eval}(\mathsf{F}, \mathcal{K}, G, M_1, i)$ and $\texttt{Eval}(\mathsf{F}, \mathcal{K}, G, M_2, i)$ are the result of two PRF_l invocations on different input values, also implying pairwise independence as well as uniformity.

DIST is either set to $\perp$ or AIU. The first case is universally valid. For the second case, we know that $\mathtt{Z}_{si}$ is chosen uniformly and independently of all other values, or no unrelated PRF node is labeled AIU and provided the same input. Thus, any other unrelated node labeled AIU is necessarily the result of a PRF invocation on a different input, guaranteeing independence. Uniformity is given by the fact that $\mathtt{Z}_{si}$ results from a PRF invocation. □

Game_3 simplifies the actual evaluation function of Game_1 by instantiating the PRF nodes with a random function (Game_1 to Game_2) and handling typing errors (Game_2 to Game_3). With CondExp() in Figure 9, we formally describe the conditions for asserting the wrong properties (i.e., a typing error). We now show that both adaptions are negligible, by providing an upper bound for the probability of distinguishing the games depending on the number of nodes $|V|$ in a graph G. Note, that we consider PRF nodes here, and TYPE = PRF must hold for the last node of a secure scheme. Later, in Theorem III.8, we use these results to give an upper bound for the security of the MACs schemes resulting from our framework.

Lemma III.4 (Correctness of game hopping)**.** Game_1 *and* Game_3 *are computationally indistinguishable for a* PPT *adversary* $\mathcal{A}$*.*

Proof. Game_1 and Game_2 differ in the setting of the *bad*-flag and in replacing the PRF with a true random function. A standard reduction shows that the advantage of an attacker in distinguishing Game_1 and Game_2 is bounded by the security bound $\mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{A},f}$ against the used PRF.

Game_2 and Game_3 are identical unless Game_3 sets the *bad*-flag and the assertions from typing fail. To calculate an upper bound for this difference, we assume an attacker $\mathcal{A}$ asking q queries to Game_3 against an arbitrary graph G. We independently look at the probability of the *bad*-flag being set at node i in query x, assuming it was not set before. Additionally, we use $|V|$ as an upper bound for the number of nodes in a graph. The *bad*-flag gets set in query v at node j with parent p if one of the following two conditions is satisfied:

$$\exists w, \mathtt{TYPE}_p \geq \mathtt{AU} \\ \wedge \mathtt{Z}_{wp} = \mathtt{Z}_{sp} \wedge \mathtt{IB}(G, M_v, j) \neq \mathtt{IB}(G, M_w, j) \quad (1)$$

$$\exists w, \mathtt{DIST}_j = \mathtt{AIU} \\ \wedge (\exists r, r \notin rel(j), \mathtt{DIST}_r \geq \mathtt{CR} \wedge \mathtt{Z}_{wr} = \mathtt{Z}_{vp}) \quad (2)$$

Condition (1) considers errors for TYPE. It applies if at least one previous message gives the same value at parent p. Since p is labeled TYPE $\geq$ AU, it is collision-resistant with probability $\leq 2^{-\lambda}$. Consequently, we can bound the probability of this case by

$$\Pr[\text{Condition (1)}] \leq \sum_{a=0}^{q-1} |V| \cdot \frac{a}{2^\lambda} \leq \frac{q^2 \cdot |V|}{2^\lambda}.$$

Condition (2) considers errors for DIST. For a PRF node, $\mathtt{DIST}_j \geq \mathtt{CR}$ implies $\mathtt{DIST}_j = \mathtt{AIU}$. Due to the partial ordering from Lemma III.1, we can bound the probability by considering collisions between unrelated nodes labeled DIST = CR. An upper bound for this probability is $x \cdot |V| \cdot 2^{-\lambda}$. For q queries, we then have

$$\Pr[\text{Condition (2)}] \leq \sum_{a=0}^{q-1} |V| \cdot (a \cdot |V|) \cdot \frac{1}{2^\lambda} \leq \frac{q^2 \cdot |V|^2}{2^\lambda}.$$

Through additive combining both bounds, we get a negligible upper bound for the difference between the games. □

E. Variable Length MAC Construction

The framework can type graphs of arbitrary length with the previously described and proven type inference. However, we do not want to type every graph first; rather, we want to be able to type an archetype to imply the typing for a similar graph with a different message length. That means, if we type a graph $G_1 \| G_2 \| G_3$, we want to be sure that $G_1 \| G_2 \| G_2 \| G_3$ and $G_1 \| G_2 \| G_2 \| \cdots \| G_3$ have the same typing.

Since the iteration graphs are deterministic, they always map to the same type tuple in OUTS for the same type tuple in INS. We call the function that describes the mapping for all possible INS to OUTS tuples a *type mapping function*. We can use this to efficiently infer the typing of the combination of iteration graphs based on the typing of those subgraphs.

Definition III.5 (Type mapping function)**.** Let $G = (V, P) \in (L^*, (\mathbb{N}^*)^*)$ be a subgraph of our framework. Let $t = (\mathtt{TYPE}, \mathtt{DIST}, \mathtt{KEYS}, \mathtt{LEN})$ be a type tuple. The *type mapping function* $\mathsf{TM}\colon G \times t \to t$ outputs the type tuple of the OUTS or OUTP node for a given type tuple in the INS node, according to the typing of G using the type inference of Section III-C.

Our framework allows storing and efficiently using the limited number of mapping rules for TM to type a concatenation of iteration graphs with desired properties. If a fixed point of the type mapping function for an iteration graph is known, then a secure concatenation is also possible without any further computation:

Lemma III.6 (Fixed points of type mapping)**.** *Let* G_2 *be an iteration graph and* TM *be a type mapping function. If* $\exists t : \mathsf{TM}(G_2, t) = t$*, then* $\mathsf{TM}(G_2 \| \ldots \| G_2, t) = t$ *holds.*

Proof. The correctness of typing follows from the determinism of the typing procedure. Since $G_2 \| \ldots \| G_2$ is an arbitrary graph with correct typing, assertions implied by the types also apply due to Theorem III.3. □

Now that we have a labeled computation graph with three subgraphs $G = G_1 \| G_2 \| G_3$, an automatically assignable type system for G and a way to extend this typing to the variable-length graph $G_1 \| G_2 \| \ldots \| G_2 \| G_3$, we can finally present the corresponding MAC construction:

Construction III.7 (MAC from subgraphs)**.** Let M be a message and let $\mathsf{F}\colon \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\lambda$ be the function that implements the PRF_l nodes. Let G_1 be an initialization graph, let G_3 be a finalization graph, and let $G_2^1, \ldots, G_2^c$ be c iteration graphs, s.t. $|M|/\lambda = |\mathtt{IB}(G, M, |V|)|$. The computation graph is $S = G_1 \| G_2^1 \| \ldots \| G_2^c \| G_3$. Then, we define the three Π_{MAC} algorithm as follows:

KGen: Sample all keys of keyset $\mathcal{K}$ uniformly random from $\{0,1\}^\lambda$.

Mac: On input a keyset $\mathcal{K}$, a message M, compute the tag with $T = \texttt{Eval}(\mathsf{F}, \mathcal{K}, S, M, |S.V|)$ and output T.

Vrfy: On input a keyset $\mathcal{K}$, a message M, and a tag T, output 1 if $T = \texttt{Eval}(\mathsf{F}, \mathcal{K}, S, M, |S.V|)$ and otherwise 0.

Theorem III.8. *Construction III.7 is a secure variable-length* Π_{MAC} *against* PPT*adversaries, if* F *is a secure PRF and* S *can be typed, s.t. the typing of* $G_1 \| G_2^1 \| \dots \| G_2^c$ *is* t *with* $t \geq \texttt{AU}$ *and for* G_3 *is* $t' = \texttt{PRF}$ *with* INS*-node(s) set to* t*.*

Proof. We base our proof on Lemma 3.2 from Mihir Bellare [7]. Essentially, it states that the correct composition of a (computational) almost universal hash function family and a PRF is a PRF. See Appendix VIII for a detailed description.

The computation of our scheme split into an almost universal function $G_1 \| G_2^1 \| \dots \| G_2^c$ and a fixed-length PRF G_3 is given by $\texttt{Eval}(\mathsf{F}, \mathcal{K}, G_3, \texttt{Eval}(\mathsf{F}, \mathcal{K}, M, S \setminus \{G_3\}, M, |S \setminus \{G_3\}.V|), |G_3.V|)$. Note that G_3 can only be typed PRF if for the last PRF_l node in G_3 with index i and for indices $j < i$ we get that $\texttt{KEYS}_j \cap \{K_l\} = \emptyset$ holds.

We apply Lemma 3.2 [7] for the security bound and use the bounds for hopping from Game_1 to Game_2 and from Game_2 to Game_3 for the typing properties from Lemma III.4. Let $\mathcal{A}$ be an adversary against S asking q queries, using instantiations of S with at most $|V|$ nodes. Then we can bound the security of S using the bound of an adversary $\mathcal{A}_f$ against G_3 and $\mathcal{A}_{f'}$ against $S \setminus \{G_3\}$ with

$$\mathsf{Adv}^{\mathrm{PRF}}_{\mathcal{A},S} \leq \mathsf{Adv}^{\mathrm{PRF}}_{\mathcal{A}_f, G_3} + \binom{q}{2} \cdot \left(\mathsf{Adv}^{au}_{\mathcal{A}_{f'}, S \setminus \{G_3\}} + \frac{q^2 \cdot |V|^2}{2^\lambda} + \frac{q^2 \cdot |V|}{2^\lambda} + \frac{1}{2^\lambda} \right).$$

Note that in the case of no collision, the computation almost universal cau-advantage is still bounded per definition by $2^{-\lambda}$, the last summand in our term. We conclude the proof by noting that any secure PRF is a secure MAC [38], since we generally assume canonical verification for deterministic MAC, and that the upper security bound is negligible in λ. □

F. Adapting the Finalization Subgraph

Initially, inspired by schemes like XCBCMAC, our framework handles keys independently from PRF nodes. However, it turned out that this resulted in no advantages for the initialization and iteration graphs; it was only relevant for the finalization graphs. Since the PRF nodes have to be keyed anyway to handle the message blocks securely, further keys have little advantage, and we have switched to representing PRF and keys using a single node. To still cover constructions that apply optimizations in the finalization graph, we present an alternative approach that utilizes the modularity of our framework.

Figure 4 gives an overview of the three constructions we consider and their adaption for the finalization graph. XCBCMAC has two additional keys and uses them depending on the padding. Instead, TMAC has one fewer key by hashing two different constants Cst_1 or Cst_2 with an additional key. OMAC does not need any further keys by utilizing the encryption of a zero block $E_{K_1}(0^n)$ and combining this with field operations.

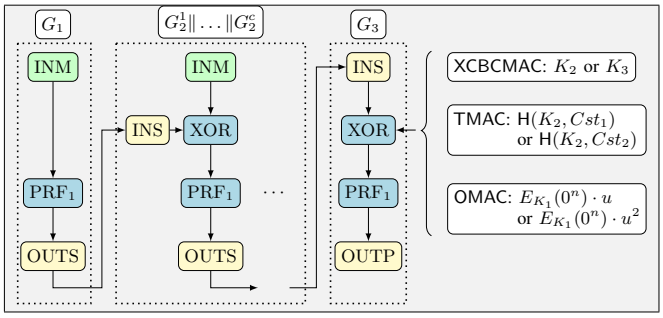

Fig. 4: Overview of the security and key optimizations for CBCMAC from XCBCMAC, TMAC, and OMAC.

For all three variants, we can utilize the modularity of our framework. Since they are based on CBCMAC and thus have an initialization and iteration subgraph with the same properties of strong universality as an assumption, we can replace the finalization graph with the adaption of the respective construction. This way, we can cover the three methods and transfer the optimization to other initialization and iteration subgraphs as long as they have comparable properties. As an example, the corresponding construction for XCBCMAC is as follows:

Construction III.9. (XCBCMAC Framework Adaption) Let $\mathsf{F}\colon \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\lambda$ be the function that implements the PRF_l nodes. Let G_1 be an initialization graph and let $G_2^1, \dots, G_2^c$ be c iteration graphs, s.t. $|M|/\lambda = |\texttt{IB}(G, M, |V|)|$. The computation graph is $S = G_1 \| G_2^1 \| \cdots \| G_2^c$. For a message M, a keyset $\mathcal{K}$ and two additional keys $K_1, K_2 \notin \mathcal{K}$, the function $\mathsf{Mac}(K, M)$ outputs

$$T = \mathsf{F}(K_i, \texttt{Eval}(\mathsf{F}, \mathcal{K}, S, M, |S.V|) \oplus K_j),$$

where $K_i \in \mathcal{K}$ and $K_j = K_1$ if padding is necessary and $K_j = K_2$ otherwise.

If the keys are sampled uniformly random, F is a secure PRF, and S is typed $t \geq \texttt{PU}$, then III.9 is a secure MAC. The security follows directly from the proof of XCBCMAC since S fulfills the necessary properties [14].

IV. Implementation

We implemented the framework in C++14 and provide the source code under an open source license[2]. This section gives an overview of the implementation and optimization. We discuss the results in Section V.

Implementation. We represent a MAC scheme with three graphs: initialization, iteration, and finalization, where each of these graphs is a list of node labels given in order of evaluation. For example, the iteration graph

[2]https://github.com/3Komma1415/AutomatedMACs

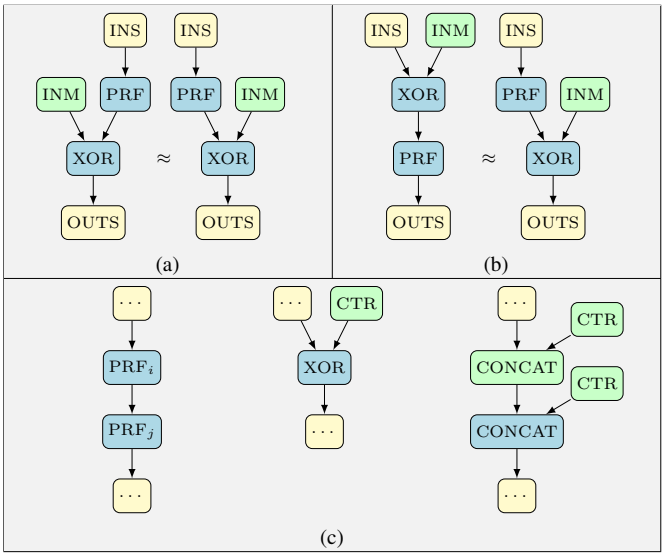

Fig. 5: (a) Example for Optimization 1, using graph isomorphism from the commutative property of XOR. (b) Example for Optimization 2 based on transition dependencies. (c) Examples for Optimizations 3.1, 3.2, and 3.3, pruning cryptographic irrelevant subgraphs.

for modified PMAC (see Figure 6) is represented by $[\mathrm{INM}, \mathrm{CTR}, \mathrm{PRF}_0, \mathrm{XOR}, \mathrm{PRF}_1, \mathrm{INS}, \mathrm{XOR}, \mathrm{OUTS}]$.

We prioritize the iteration graphs and generate all possible node label combinations and then all corresponding graphs using permutation. Consequently, our approach exhaustively analyzes all graphs of a given size. Then, we initially validate each graph, containing an input-output degree checkup, checking if the typing of the subgraph does not abort for input $(\mathtt{TYPE}, \mathtt{DIST}) = (\mathtt{PU}, \mathtt{AIU})$ (strongest type for the initialization subgraph), and testing for a type mapping fixed point. Afterward, given the reduced cryptographic relevance, we only search for at least one initialization and finalization graph so that the resulting scheme is secure. We further evaluate whether the results are parallelizable by analyzing the iteration graph. We consider schemes parallelizable if there is no path from a previous computation (INS node) to a PRF.

The restrictions for the node label combinations are as follows: Any list must start with a INM, INS, or CTR-node and end with either OUTS or OUTP. Furthermore, the implementation supports only four differently keyed PRF, which are hard coded. This arises from the runtime increase as the number of nodes grows. An upper bound on the time complexity using Landau notation is $\mathcal{O}([(n+m-4)!/((n-3)!(m-1)!)]!)$, where n is the number of nodes which can get m different labels. Here, the initial fixed nodes have already been accounted for. We have additional optimizations to improve performance and prevent duplicates, described below.

Optimization 1. Since XOR is commutative, switching the input order (see Figure 5a) does not imply different results. Thus, we compute a 64-bit hash of the subgraphs. Subsequently, we require the hash of the left subgraph to be smaller than the hash of the right subgraph. Due to the time-complexity, we consider collisions on the hash values negligible. However, two subgraphs with matching INM or CTR nodes in their parent subgraphs imply different outputs for the XOR node. Swapping them alters the evaluation order. Thus, we intentionally do not prune on these graphs.

Optimization 2. We improve the transition from a subgraph or iteration to the next. Consider Figure 5b to get an intuition. Although the right graph appears different at first, it simply relocates the PRF node from the output to the input. This similarity becomes evident with multiple subgraph iterations, where OUTS equals INS. Given our focus on distinct iteration subgraphs, we treat these two schemes as identical, as the difference, if any, lies in handling the initial and final nodes. We implemented the pruning by ensuring that the first operation on the input state is dependent on an INM node in the current subgraph. Regarding Figure 5, this results in pruning the right graph, as calling PRF directly on INS lacks further dependencies.

Optimization 3. Many subgraphs may differ significantly but remain uninteresting from a cryptographic perspective. Figure 5c shows the three variants we prune in our implementation: 3.1. Chains of PRF, since this does not result in any advantage. 3.2. Computing XOR with a CTR and a subgraph since it causes no significant impact within our framework. 3.3. Repeated appending or prepending of counter values since security advantages are already achieved by attaching only one counter.

Parallelizability. Besides the automatic synthesis of secure MACs, our framework enables an automatic search for constructions with certain properties. We demonstrate this by analyzing the results considering parallelizability. We base this approach on the design of parallel MACs such as PMAC and LightMAC. The idea is that the PRF invocations for the messages blocks are the bottleneck, which should be calculated independently so that only the results must be combined. Thus, we consider a scheme parallelizable if the iteration graph has no path from an INS-node to a PRF node. This can be determined efficiently by iterating over the results of the framework.

V. Evaluation

This section explains design decisions and compares them with prior work. Then, we consider the overall expressiveness of our approach and evaluate the associated experimental results, including the description of alternative parallelizable constructions for practical use.

A. Design Choices and Comparison

Since the idea of typed graphs inspires our framework, comparing it with the work of Malozemoff et al. [29] and Hoang et al. [30] is reasonable. In Table II, we compare the type systems, which we have significantly expanded and discuss in detail below.

A similarity between the work of Malozemoff et al. [29], and ours is the set of node labels representing the possible

Enc. [29]	Auth. Enc. [30]	MACs
$\texttt{TYPE} \in \{R, U, \bot\}$ $fam \subset \mathbb{N}$ $flags \in \{0,1\}^2$	$\texttt{TYPE} \in \{R, 1, 0, \bot\}$ $ctr \in \mathbb{N}$	$\texttt{TYPE} \in \{\bot, \texttt{CONST}, \texttt{AU}, \texttt{PU}, \texttt{PRF}\}$ $\texttt{DIST} \in \{\bot, \texttt{CR}, \texttt{AIU}\}$ $\texttt{KEYS} \subseteq \mathbb{N}$ $\texttt{LEN} \in \mathbb{N}$

TABLE II: Overview of the type system comparison of graph-based approaches for the automated analysis and synthesis considering encryption Enc., authenticated encryption Auth. Enc., and message authentication codes MACs.

computations. The follow-up work by Hoang et al. [30] on authenticated encryption reduces the number of node labels and relies on tweakable block ciphers instead.

We model counter values CTR as node labels (similarly to the extension in the Appendix of Malozemoff et al. [29]) and extend the set of possible labels with a CONCAT label for concatenation. Both adaptions allow the coverage of parallelizeable counter-sum-based schemes like LightMAC [37]. However, concatenating requires handling variable block lengths, which we therefore do not constrain. To ensure correct constructions, we expand the type system by LEN for the length of the blocks and allow the PRF node to process inputs of any length. Beyond that, we increase flexibility with the modular tripartite graph approach and by not restricting the number of input message blocks per iteration.

We initially considered an explicit key-labeled nodes approach. The idea arose from schemes like XCBCMAC. However, this did not result in any advantage for initialization and iteration graphs. Generally, we need a keyed PRF-node to process the message blocks, making further keys obsolete. Hence, we decided to consider the PRF nodes as generally keyed and use the modularity of our approach to cover schemas like XCBCMAC as Section III-F describes.

Malozemoff et al. [29] uses a TYPE-field with three properties: random R, unique U, and adversarial controlled $\bot$. Additionally, they use a family-type fam to detect related edges and bit-vector $flags$ to check whether an edge can be used as input to OUTS or PRF. The type system by Hoang et al. [30] has four properties: random R, arbitrary output $\bot$. 0 and 1 are used to reason about a node for different decryption queries. Additionally, they use a ctr type to reason about the independence of R.

In both cases, the prior work uses randomness. However, MACs are commonly deterministic. Thus, a type system for MACs needs to be enhanced with a more detailed analysis since we cannot rely on the randomness properties in the security analysis. Besides the implications from the MAC security experiment Section III-B describes, this is another reason for an advanced type system and the distinction between TYPE and DIST. Since most MACs use multiple keys, we also introduced the KEYS label to check for key independence.

B. Expressiveness of the Framework

This section examines the common MAC constructions (We refer the reader to the appendix of the full version for a comprehensive summary of MAC constructions), indicating, on the one hand, whether and how our framework covers them and, on the other hand, the reasons why they are not covered.

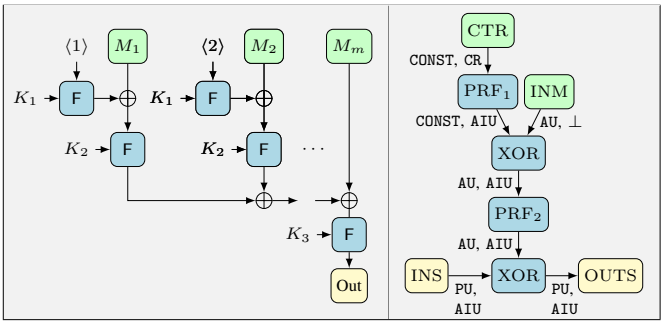

Fig. 6: The left visualises the simplified PMAC variant. The right shows the corresponding iteration graph, including the typing.

Representable Schemes. CBCMAC can be represented by the framework and is correctly identified as insecure. The variants ECBCMAC, FCBCMAC, and TMAC pass the typing check and are correctly identified as secure. Since our framework utilizes a counter and concatenation node, it also covers LightMAC.

Utilizing the modularity approach from Section III-F, our framework also covers XCBCMAC, TMAC, and OMAC and the associated optimizations.

NMAC is a more general description of MACs, which our framework can represent in different ways. This also covers the structure of HMAC. However, the framework can only prove the security of HMAC in an idealized setup, assuming that two independent keys are used. Nevertheless, the strength of HMAC is that the security can be proven with only one key and much weaker assumptions than our framework has. Our general methodology for comprehensively generating secure MACs does not allow the same level of detail inherent with manual analysis, since the individual techniques differ greatly.

Regarding PMAC, we need to differentiate. Generally, the framework does not include PMAC due to missing field operations. However, those are only used for optimizations. The following simplified version of PMAC is included: We omit the whitening of the last block by XORing with $x^{-1} \cdot L$, which is redundant when dealing with messages of multiple block sizes. Additionally, we use three differently keyed PRF instead of one. One PRF for replacing $\gamma_i \cdot L$ with $\mathsf{PRF}(K_1, \langle i \rangle)$, one for the intermediate result and the message block, and the last for the finalization block. Figure 6 shows this simplified PMAC and its relation to our framework.

Not Representable Schemes. We deliberately exclude all constructions within the Carter-Wegman paradigm. Simplified, the Carter-Wegman Mac is defined as follows:

$$R \leftarrow\!\!\$\ \{0,1\}^\lambda$$
$$T \leftarrow \mathsf{H}(K_1, M) \oplus \mathsf{F}(K_2, R)$$

where H is a hash function, F is a PRF, K_1 and K_2 are keys, and the output is (T, R) for input message M. The first step

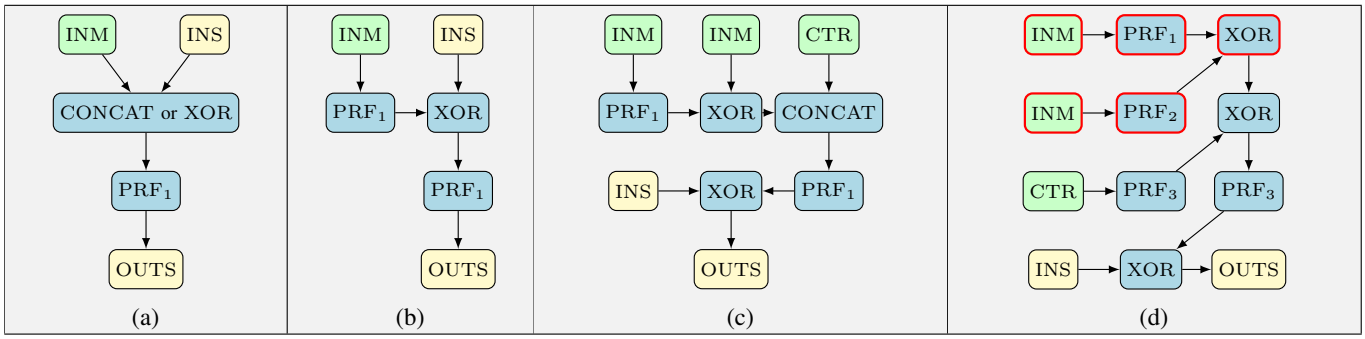

Fig. 7: Shown are five generated iteration graphs: (a) are the two smallest secure but not parallelizable schemes, where the XOR variant resembles ECBCMAC. (b) shows a slightly worse adaption of (a), where the left PRF node is not necessary. (c) shows one of the parallelizable hybrid schemes. (d) presents a parallelizable hybrid with combiner properties for the first XOR node.

Nodes	1 PRF		2 PRF		3 PRF	
	S	P	S	P	S	P
5	2	0	0	0	0	0
6	1	0	2	0	0	0
7	11	1	0	0	0	0
8	37	3	76	8	0	0
9	85	4	186	10	192	12
10	350	22	806	50	312	0
11	1066	63	4940	394	5118	480
12	3245	148	/	/	/	/

TABLE III: Summary data from framework computation, including the number of secure schemes (S) and the size of the parallelizable subset (P) considering iteration graphs with one, two or three differently keyed PRF.

involves sampling a random value R, resulting in a randomized construction. Since R is derived from a PRF, true randomness is not required. It is sufficient that R is unique, serving as a nonce (a number only used once). However, choosing R as a counter would result in a stateful Mac algorithm. Both cases fall outside the scope of our framework. Nonetheless, the framework's modularity allows construction to address this, similar to the approach Section III-F describes. While these constructions offer interesting insights into various cryptographic topics, they extend beyond the focus of this work. Consequently, our framework does not cover schemes like XORMAC, VMAC, and UMAC.

The framework does not cover more schemes whose design and proof technique strongly differ from those previously mentioned. This concerns SpongeMAC based on sponge functions and includes one-time MAC constructions. Notably, Poly1305 is excluded as its universal hash function does not satisfy the necessary properties for the PRF node. Existing approaches for Poly1305 are either one-time or Carter-Wegman-based, and adapting the framework to include these, as already described above, falls outside the scope of this paper.

C. Results from Implementation

We now present the computational results of our implementation, which Section IV describes. Table III shows the results with different constraints. We only consider the iteration graphs since the set of optimal iteration and finalization graphs is limited and well-studied.

For each discovered secure MAC scheme, our implementation outputs additional information for each iteration graph: the number of PRF nodes, the total sum of bits given to a PRF, and the ratio of input bits (number of INM-nodes times bits per block) to PRF input bits. Combined with the number of computational nodes, we base our theoretical performance approximation on this information.

All common constructions in the scope of the framework's expressiveness can be represented with at most 12 nodes. For instance, Figure 7a resembles the ECBCMAC construction. This also includes the modified PMAC scheme displayed in Figure 6.

Considering simple constructions without additional properties besides security, we found no new schemes with comparable efficiency. Figure 7b shows the reason using an example: The size of the graph instances for these methods is sufficiently small that all efficient constructions have already been found, and further nodes reduce efficiency.

However, the strength of our methodology emerges when considering MACs with properties beyond security. Our framework allows for hybrid iteration graphs, i.e., a subgraph that handles multiple message blocks. In particular, the parallel hybrid schemes appear useful. In terms of the number of nodes and the message block per PRF ratio, many of these schemes are theoretically comparable in efficiency to existing parallel schemes. Among the many hybrid schemes generated, those composed of common patterns seem intuitive and straightforward. Figure 7c shows an example of such a scheme with 10 nodes for the iteration graph. Basically, this is a combination of CBCMAC and LightMAC, in which each iteration graph processes a larger number of message blocks. Note that this subset of hybrid schemes emerges directly as a result from our implementation, but also from the type mapping function of Definition III.5. The same schemes can be obtained by concatenating the subgraphs of CBCMAC and LightMAC using the fixed points of their type mapping functions.

In theory, the efficiency of such hybrid variants is slightly better than that of LightMAC since operations like CTR and

CONCAT are minimized. Admittedly, this is not a major factor, but it is significant when scaling up to large data sets.

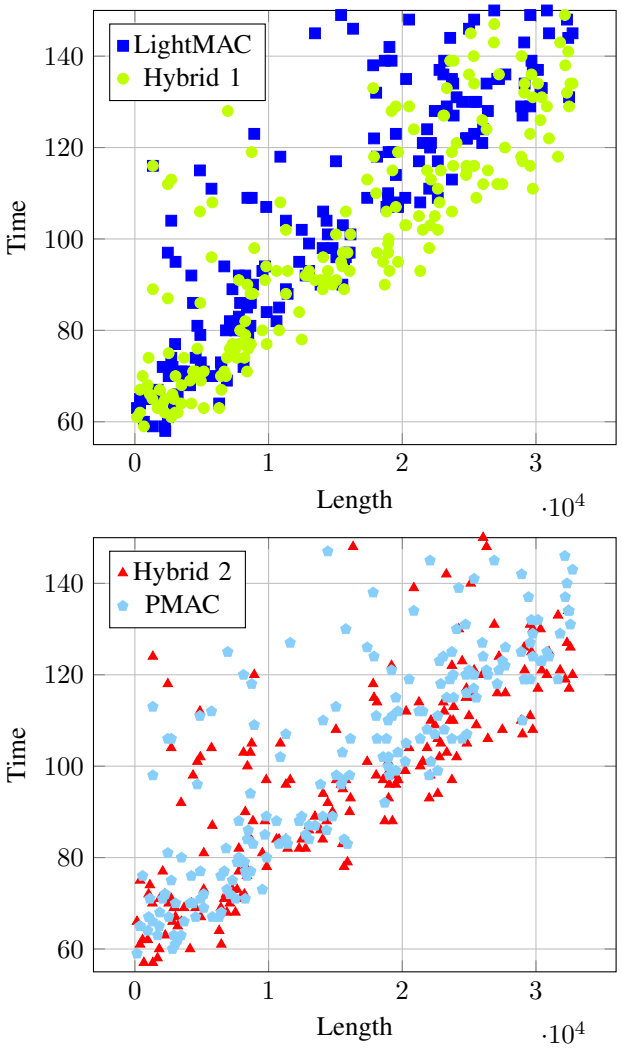

Fig. 8: Results of the performance comparison between our parallelizable hybrid schemes and comparable implementations of LightMAC and PMAC. An Apple M2 chip and 16GB of RAM were used for our computational tests. We measured the computation time in microseconds for messages with up to 4096 blocks, each of 64 bits in size.

We implemented two of the parallelizable hybrid schemes. Hybrid 1 implements Figure 7c, and Hybrid 2 is a variant, where the CBCMAC part digests four instead of 2 message blocks. Additionally, we implemented comparable implementations for the common existing MAC schemes LightMAC [37] and PMAC [18]. For PMAC we use the variant output by our framework described in Figure 6. Note that in practice, existing implementations of CBCMAC variants, LightMAC, and PMAC can be combined to simplify implementation complexity.

The administrative part for managing the threads is the same for all four implementations. Since creating threads reduces efficiency, we do not have a thread for every iteration graph but divide the load evenly between a fixed number of threads.

The standard comparison approach is comparing the clock cycles per iteration graph. However, since the size of the iteration graphs varies and larger graphs implicitly require more clock cycles, we use a method based on the time measurement. We randomly test for messages with up to 4096 message blocks with 64-bit block sizes. We instantiate the PRF using AES encryption from OpenSSL [39]. Figure 8 shows the result.

The results highlight that the hybrid parallel schemes generated by our framework have comparable efficiency to existing methods. In some cases, especially with growing message length, our schemes are slightly more efficient due to the correlation to the amount of data per iteration block. However, this relationship may not generally apply, as hardware restrictions and optimization significantly affect parallel computation. Therefore, we do not consider the hybrid parallel schemes of our framework as a replacement for LightMAC or PMAC, but rather as alternatives or extensions for better efficiency optimization.

Our framework allows graphs with arbitrary size and any number of keys. Intuitively, this leads to redundancy in most graphs, which impairs efficiency. Nevertheless, redundancy can be practically relevant: A cryptographic *Combiner* [40], [41] combines two functions into a failure-tolerant function such that the combination remains secure as long as one of the two functions is secure. Considering a PRF F, the common construction for a Combiner is $\mathsf{Comb}((K_1, K_2), M) = \mathsf{F}(K_1, M) \oplus \mathsf{F}(K_2, M)$ with independent and uniformly random keys K_1 and K_2. Combiners provide robustness for standards such as TLS [42]. Especially the results with three or more differently keyed PRFs could be interesting for future research in this area. Figure 7d shows a parallelizable hybrid scheme with 12 nodes. Since we have few constraints, INM nodes can represent the same message block. The nodes outlined in red represent the combiner.

VI. Acknowledgments

We gratefully acknowledge the anonymous reviewers for their invaluable comments and suggestions. This work was partially supported by Deutsche Forschungsgemeinschaft as part of the Research and Training Group 2475 “Cybercrime and Forensic Computing” (grant number 393541319/GRK2475/1-2019), grant 442893093, by the state of Bavaria at the Nuremberg Campus of Technology (NCT) which is a research cooperation between the Friedrich-Alexander- Universität Erlangen-Nürnberg (FAU) and the Technische Hochschule Nürnberg Georg Simon Ohm (THN), and by the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union’s Horizon Europe research and innovation program in the scope of the CONFIDENTIAL6G project under Grant Agreement 101096435. The contents of this publication are the sole responsibility of the authors and do not in any way reflect the views of the EU.

References

[1] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2000, pp. 531–545.

[2] T. Dierks and C. Allen, *RFC 2246 - The TLS Protocol Version 1.0*, Internet Activities Board, Jan. 1999.

[3] S. Kent and K. Seo, "Security Architecture for the Internet Protocol," Request for Comments: 4301, RFC Editor, RFC 4301, December 2005. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4301.txt

[4] T. Ylonen and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture," Request for Comments: 4251, RFC Editor, RFC 4251, January 2006. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4251.txt

[5] E. Petrank and C. Rackoff, "Cbc mac for real-time data sources," *Journal of Cryptology*, vol. 13, no. 3, pp. 315–338, 2000.

[6] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *Crypto*, vol. 96. Springer, 1996, pp. 1–15.

[7] M. Bellare, "New proofs for nmac and hmac: Security without collision-resistance," in *Annual International Cryptology Conference*. Springer, 2006, pp. 602–619.

[8] J. M. Turner, "The keyed-hash message authentication code (hmac)," *Federal Information Processing Standards Publication*, vol. 198, no. 1, pp. 1–13, 2008.

[9] B. Guido, D. Joan, P. Michaël, and V. Gilles, "Cryptographic sponge functions," 2011.

[10] M. Dworkin, "Recommendation for block cipher modes of operation," *NIST special publication*, vol. 800, p. 38G, 2016.

[11] J. Kelsey, S.-j. Chang, and R. Perlner, "Sha-3 derived functions: cshake, kmac, tuplehash, and parallelhash," *NIST special publication*, vol. 800, p. 185, 2016.

[12] B. Preneel and P. C. Van Oorschot, "Mdx-mac and building fast macs from hash functions," in *Annual International Cryptology Conference*. Springer, 1995, pp. 1–14.

[13] M. Bellare, K. Pietrzak, and P. Rogaway, "Improved security analyses for cbc macs," in *Crypto*, vol. 3621. Springer, 2005, pp. 527–545.

[14] J. Black and P. Rogaway, "Cbc macs for arbitrary-length messages: The three-key constructions," in *Advances in Cryptology—CRYPTO 2000*. Springer, 2000, pp. 197–215.

[15] K. Kurosawa and T. Iwata, "Tmac: Two-key cbc mac," in *Topics in Cryptology—CT-RSA 2003: The Cryptographers' Track at the RSA Conference 2003 San Francisco, CA, USA, April 13–17, 2003 Proceedings*. Springer, 2003, pp. 33–49.

[16] T. Iwata and K. Kurosawa, "Omac: One-key cbc mac," in *FSE*, vol. 2887. Springer, 2003, pp. 129–153.

[17] V. T. Hoang, J. Katz, and A. J. Malozemoff, "Automated analysis and synthesis of authenticated encryption schemes," in *ACM CCS 2015*, I. Ray, N. Li, and C. Kruegel, Eds. ACM Press, Oct. 2015, pp. 84–95.

[18] P. Rogaway and J. Black, "Pmac: A parallelizable message authentication code," *Preliminary Draft, October*, vol. 16, 2000.

[19] G. Barthe, J. M. Crespo, B. Grégoire, C. Kunz, Y. Lakhnech, B. Schmidt, and S. Zanella Béguelin, "Fully automated analysis of padding-based encryption in the computational model," in *ACM CCS 2013*, A.-R. Sadeghi, V. D. Gligor, and M. Yung, Eds. ACM Press, Nov. 2013, pp. 1247–1260.

[20] A. Tiwari, A. Gascón, and B. Dutertre, "Program synthesis using dual interpretation," in *International Conference on Automated Deduction*. Springer, 2015, pp. 482–497.

[21] G. Barthe, E. Fagerholm, D. Fiore, J. C. Mitchell, A. Scedrov, and B. Schmidt, "Automated analysis of cryptographic assumptions in generic group models," in *CRYPTO 2014, Part I*, ser. LNCS, J. A. Garay and R. Gennaro, Eds., vol. 8616. Springer, Heidelberg, Aug. 2014, pp. 95–112.

[22] G. Barthe, E. Fagerholm, D. Fiore, A. Scedrov, B. Schmidt, and M. Tibouchi, "Strongly-optimal structure preserving signatures from type II pairings: Synthesis and lower bounds," in *PKC 2015*, ser. LNCS, J. Katz, Ed., vol. 9020. Springer, Heidelberg, Mar. / Apr. 2015, pp. 355–376.

[23] J. A. Akinyele, M. Green, S. Hohenberger, and M. W. Pagano, "Machine-generated algorithms, proofs and software for the batch verification of digital signature schemes," in *ACM CCS 2012*, T. Yu, G. Danezis, and V. D. Gligor, Eds. ACM Press, Oct. 2012, pp. 474–487.

[24] M. Gagné, P. Lafourcade, and Y. Lakhnech, "Automated security proofs for message authentication codes," Tech. Rep.

[25] B. Carmer and M. Rosulek, "Linicrypt: A model for practical cryptography," in *CRYPTO 2016, Part III*, ser. LNCS, M. Robshaw and J. Katz, Eds., vol. 9816. Springer, Heidelberg, Aug. 2016, pp. 416–445.

[26] I. McQuoid, T. Swope, and M. Rosulek, "Characterizing collision and second-preimage resistance in linicrypt," in *TCC 2019, Part I*, ser. LNCS, D. Hofheinz and A. Rosen, Eds., vol. 11891. Springer, Heidelberg, Dec. 2019, pp. 451–470.

[27] Z. Javar and B. M. Kapron, "Preimage awareness in linicrypt," in *CSF 2023 Computer Security Foundations Symposium*. IEEE Computer Society Press, Jul. 2023, pp. 33–42.

[28] T. Hollenberg, M. Rosulek, and L. Roy, "A complete characterization of security for linicrypt block cipher modes," in *CSF 2022 Computer Security Foundations Symposium*. IEEE Computer Society Press, Aug. 2022, pp. 439–454.

[29] A. J. Malozemoff, J. Katz, and M. D. Green, "Automated analysis and synthesis of block-cipher modes of operation," in *Computer Security Foundations Symposium (CSF), 2014 IEEE 27th*. IEEE, 2014, pp. 140–152.

[30] V. T. Hoang, J. Katz, and A. J. Malozemoff, "Automated analysis and synthesis of authenticated encryption schemes," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 84–95.

[31] M. Bellare and P. Rogaway, "The security of triple encryption and a framework for code-based game-playing proofs," in *EUROCRYPT 2006*, ser. LNCS, S. Vaudenay, Ed., vol. 4004. Springer, Heidelberg, May / Jun. 2006, pp. 409–426.

[32] J. Katz and Y. Lindell, *Introduction to modern cryptography*. CRC press, 2014.

[33] M. Bellare and P. Rogaway, "The security of triple encryption and a framework for code-based game-playing proofs," in *Proceedings of the 24th Annual International Conference on The Theory and Applications of Cryptographic Techniques*, ser. EUROCRYPT'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 409–426. [Online]. Available: http://dx.doi.org/10.1007/11761679_25

[34] J. L. Carter and M. N. Wegman, "Universal classes of hash functions," *Journal of computer and system sciences*, vol. 18, no. 2, pp. 143–154, 1979.

[35] M. N. Wegman and J. L. Carter, "New hash functions and their use in authentication and set equality," *Journal of computer and system sciences*, vol. 22, no. 3, pp. 265–279, 1981.

[36] D. R. Stinson, "Universal hashing and authentication codes," in *Annual International Cryptology Conference*. Springer, 1991, pp. 74–85.

[37] A. Luykx, B. Preneel, E. Tischhauser, and K. Yasuda, "A mac mode for lightweight block ciphers," in *International Conference on Fast Software Encryption*. Springer, 2016, pp. 43–59.

[38] M. Bellare, O. Goldreich, and A. Mityagin, "The power of verification queries in message authentication and authenticated encryption," Cryptology ePrint Archive, Report 2004/309, 2004, https://eprint.iacr.org/2004/309.

[39] OpenSSL Project, "OpenSSL Documentation: The OpenSSL crypto library," https://www.openssl.org/docs/, 2024.

[40] M. Nandi and D. R. Stinson, "Multicollision attacks on some generalized sequential hash functions," Cryptology ePrint Archive, Paper 2006/055, 2006, https://eprint.iacr.org/2006/055. [Online]. Available: https://eprint.iacr.org/2006/055

[41] M. Fischlin and A. Lehmann, "Security-amplifying combiners for collision-resistant hash functions," in *Annual International Cryptology Conference*. Springer, 2007, pp. 224–243.

[42] M. Fischlin, A. Lehmann, and D. Wagner, "Hash function combiners in tls and ssl," in *Cryptographers' Track at the RSA Conference*. Springer, 2010, pp. 268–283.

VII. Technical Details of the Type Inference

We use a type inference procedure to automatically assign the types of the type system to the nodes of a labeled graph. This typing algorithm analyzes all nodes and computes their type depending on the types assigned to their predecessors if any exist. We assume the nodes are visited in some topological ordering and that the types of the predecessors have already

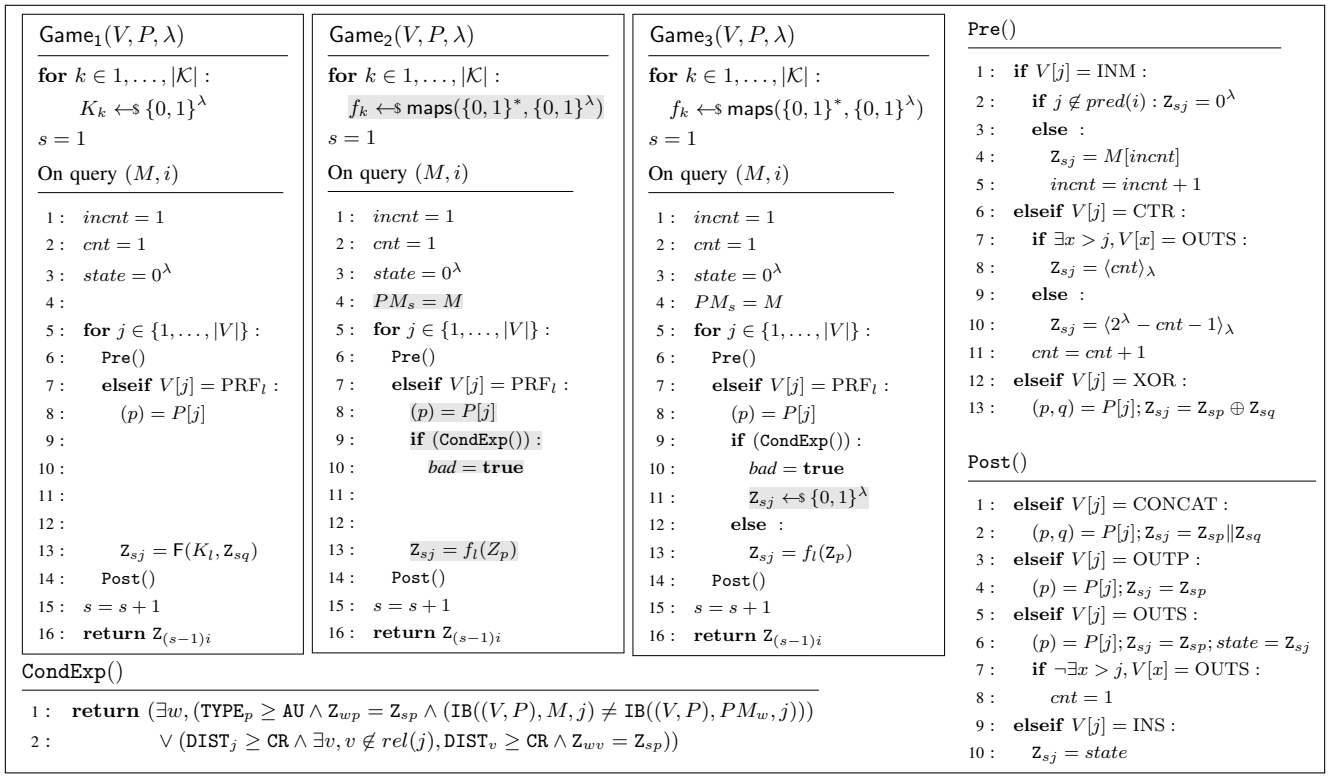

Fig. 9: Pseudocode of Game_1, Game_2 and Game_3, which are used for the proof of correct typing and security of the resulting MAC scheme. For the outsourcing of redundant code, we assume that Pre(), Post(), and CondExp() are within the scope in which they are called and can access all variables within it.

been determined. In the following, we use i to denote any node in any graph and j and k to denote its parents if they exist:

$V(i) =$ INM: We set $\mathtt{TYPE}_i = \mathtt{AU}, \mathtt{DIST}_i = \perp, \mathtt{KEYS}_i = \emptyset$, and $\mathtt{LEN}_i = \lambda$.

$V(i) =$ CTR: We set $\mathtt{TYPE}_i = \mathtt{CONST}, \mathtt{DIST}_i = \mathtt{CR}, \mathtt{KEYS}_i = \emptyset$, and $\mathtt{LEN}_i = \lambda$.

$V(i) =$ CONCAT: We set $\mathtt{TYPE}_i = \perp$ if $\mathtt{TYPE}_j = \mathtt{TYPE}_k = \perp$ or $\mathtt{TYPE}_j = \perp \wedge \mathtt{TYPE}_k = \mathtt{CONST}$ or $\mathtt{TYPE}_k = \perp \wedge \mathtt{TYPE}_j = \mathtt{CONST}$. If $\mathtt{TYPE}_j = \mathtt{TYPE}_k = \mathtt{CONST}$ we set $\mathtt{TYPE}_i = \mathtt{CONST}$. Otherwise, $\mathtt{TYPE}_i = \mathtt{AU}$.

We set $\mathtt{DIST}_i = \perp$ if $\mathtt{DIST}_j = \mathtt{DIST}_k = \perp$, otherwise $\mathtt{DIST}_i = \mathtt{CR}$. The following tables provide an overview for assigning $\mathtt{TYPE}_i$ and $\mathtt{DIST}_i$ based on parent nodes j (row) and k (column):

TYPE	$\perp$	CONST	AU	PU	PRF
$\perp$	$\perp$	$\perp$	AU	AU	AU
CONST	$\perp$	CONST	AU	AU	AU
AU	AU	AU	AU	AU	AU
PU	AU	AU	AU	AU	AU
PRF	AU	AU	AU	AU	AU

DIST	$\perp$	CR	AIU
$\perp$	$\perp$	CR	CR
CR	CR	CR	CR
AIU	CR	CR	CR

Moreover, $\mathtt{KEYS}_i = \mathtt{KEYS}_j \cup \mathtt{KEYS}_k$ and $\mathtt{LEN}_i = \mathtt{LEN}_j + \mathtt{LEN}_k$.

$V(i) =$ XOR: We use $\mathtt{Comb}(t_j, t_k)$, which is shown in Figure 10, to compute the type of node i depending on the type t_j of node j and t_k of node k. If Comb outputs $\perp$, we abort further labeling.

$V(i) = \text{PRF}_l$: If $\mathtt{TYPE}_j = \perp$ we set $\mathtt{TYPE}_i = \perp$. If $\mathtt{TYPE}_j = \mathtt{CONST}$ we set $\mathtt{TYPE}_i = \mathtt{CONST}$. If $\mathtt{TYPE}_j \geq \mathtt{AU}$ we set $\mathtt{TYPE}_i = \mathtt{PRF}$ if $\{K_l\} \cap \mathtt{KEYS}_j = \emptyset$ and $\mathtt{TYPE}_i = \mathtt{PU}$ otherwise.

Regarding $\mathtt{DIST}_i$, we set $\mathtt{DIST}_i = \mathtt{AIU}$ if $\mathtt{DIST}_j \neq \perp$ and $\mathtt{DIST}_i = \perp$ otherwise. Finally, we set $\mathtt{KEYS}_i = \{K_l\} \cup \mathtt{KEYS}_j$, and $\mathtt{LEN}_i = \lambda$.

$V(i) =$ OUTS: We assign node i the same type tuple as parent node j. If $\exists r, V(r) = \text{INS} \wedge \mathtt{LEN}_r \neq \mathtt{LEN}_j$, we abort further labeling, as we require the iteration function not to modify the length of the state carried over to the next iteration.

$V(i) =$ INS: We set the same typing as the OUTS node from the previous subgraph. Together with OUTS, this enables the concentration of subgraphs.

$V(i) =$ OUTP: If $\mathtt{TYPE}_j = \mathtt{PRF}$ we accept the labeling. Otherwise, we abort further labeling if necessary.

If the type inference of a subgraph ends with an abort, we assume no typing that would lead to a secure variable-length MAC scheme can be assigned to the graph.

VIII. The PRF(cAU) = PRF Lemma

The proof and the security bound for the results of our framework use Lemma 3 of the NMAC and HMAC paper from Mihir Bellare [7, Lemma 3.2]. We now restate the Lemma for convenience.

The composition of families $h\colon \{0,1\}^b \times \{0,1\}^c \to \{0,1\}^c$ and $F\colon \{0,1\}^k \times D \to \{0,1\}^b$ is the family $hF\colon \{0,1\}^{c+k} \times D \to \{0,1\}^c$ defined by $hF(K_{\text{out}} \| K_{\text{in}}, M) =$

```
Comb(t_j, t_k)
1 : // Let t_i = (TYPE_i, DIST_i, KEYS_i, LEN_i) be the result's type tuple
2 : if (LEN_j ≠ LEN_k) : return ⊥
3 : if (TYPE_j = CONST) : TYPE_i = TYPE_k
4 : if (TYPE_k = CONST) : TYPE_i = TYPE_j
5 : if ((DIST_j = AIU ∧ DIST_k = AIU) ∨ (KEYS_j ∩ KEYS_k = ∅)) :
6 :   if ((TYPE_j = (PU ∨ PRF) ∧ TYPE_k = (PU ∨ PRF)) :
7 :     TYPE_i = PU
8 :   elseif (TYPE_j = (PU ∨ PRF) ∨ TYPE_k = (PU ∨ PRF)) :
9 :     TYPE_i = AU
10 :  else : TYPE_i = ⊥
11 : else : TYPE_i = ⊥
12 : if (DIST_j = AIU ∧ DIST_k = AIU) : DIST_i = AIU
13 : elseif ((DIST_j = AIU ∨ DIST_k = AIU) ∧ (KEYS_j ∩ KEYS_k = ∅)) :
14 :   DIST_i = AIU
15 : else : return ⊥
16 : KEYS_i = KEYS_j ∪ KEYS_k
17 : LEN_i = LEN_j
18 : return t_i
```

Fig. 10: Pseudocode for the type inference of an XOR node based on parent node's types.

$h(K_{\text{out}}, F(K_{\text{in}}, M))$. The following lemma says that if h is a PRF (pseudorandom function) and F is cAU (computational almost universal) then hF is a PRF [7].

Lemma VIII.1. *(PRF (cAU) = PRF) [7] Let $B = \{0,1\}^b$. Let $h\colon \{0,1\}^c \times B \to \{0,1\}^c$ and $F\colon \{0,1\}^k \times D \to B$ be families of functions, and let $hF\colon \{0,1\}^{c+k} \times D \to \{0,1\}^c$ be defined by*

$$hF(K_{out} \| K_{in}, M) = h(K_{out}, F(K_{in}, M))$$

for all $K_{out} \in \{0,1\}^c, K_{in} \in \{0,1\}^k$ and $M \in D$. Let $\mathcal{A}_{hF}$ be a PRF-adversary against hF that makes at most $q \geq 2$ oracle queries, each of length at most n, and has time complexity at most t. Then there exists a PRF-adversary $\mathcal{A}_h$ against h and an au-adversary A_F against F such that

$$\mathsf{Adv}^{PRF}_{hF}(\mathcal{A}_{hF}) \leq \mathsf{Adv}^{PRF}_{h}(\mathcal{A}_h) + \binom{q}{2} \cdot \mathsf{Adv}^{au}_{F}(\mathcal{A}_F).$$