Check for updates

ProMACs: Progressive and Resynchronizing MACs for Continuous Efficient Authentication of Message Streams

Frederik Armknecht*
University of Mannheim
Germany
armknecht@uni-mannheim.de

Paul Walther*
TU Dresden
Germany
paul.walther@tu-dresden.de

Gene Tsudik
UC Irvine
USA
gts@ics.uci.edu

Martin Beck
TU Dresden
Germany
martin.beck1@tu-dresden.de

Thorsten Strufe
KIT and CeTI TU Dresden
Germany
strufe@kit.edu

ABSTRACT

Efficiently integrity verification of received data requires Message Authentication Code (MAC) tags. However, while security calls for rather long tags, in many scenarios this contradicts other requirements. Examples are strict delay requirements (e.g., robot or drone control) or resource-scarce settings (e.g., LoRaWAN networks with limited battery capacity).

Prior techniques suggested truncation of MAC tags, thus trading off linear performance gain for exponential security loss. To achieve security of full-length MACs with short(er) tags, we introduce Progressive MACs (ProMACs) – a scheme that uses internal state to gradually increase security upon reception of subsequent messages. We provide a formal framework and propose a provably secure, generic construction called Whips. We evaluate applicability of ProMACs in several realistic scenarios and demonstrate example settings where ProMACs can be used as a drop-in replacement for traditional MACs.

CCS CONCEPTS

• **Security and privacy** → **Hash functions and message authentication codes**; **Formal security models**; *Domain-specific security and privacy architectures*; • **Theory of computation** → *Cryptographic primitives*.

KEYWORDS

Message Authentication Codes, Stream Authentication, Progressing Security, Sensor Networks, Drone Control, Robot Control

ACM Reference Format:
Frederik Armknecht, Paul Walther, Gene Tsudik, Martin Beck, and Thorsten Strufe. 2020. ProMACs: Progressive and Resynchronizing MACs for Continuous Efficient Authentication of Message Streams. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer andCommunications Security (CCS '20), November 9–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3372297.3423349

*Both authors contributed equally to the paper

CCS '20, November 9–13,2020, Virtual Event, USA

ACM ISBN 978-1-4503-7089-9/20/11…$15.00
https://doi.org/10.1145/3372297.3423349

1 INTRODUCTION

Verifying authenticity and integrity of received data (e.g., a packet stream) is usually achieved by appending a Message Authentication Code (MAC) tag to each packet. Increasing the tag length linearly yields exponential security gains (up to the key size). The use of long(er) per packet MAC tags is typically justified by high (and growing) network speeds and large packet sizes in many application domains.

However, some recent and emerging communication settings impose ultra-low latency requirements and involve low-power devices, which makes the bandwidth overhead of long MAC tags to be too high. Examples include: the Internet of Things (IoT) with small battery-operated devices, vehicular communication, as well as robot and drone control. Communication in these examples is characterized by very high frequency streams of very small control packets that are subject to very strict timing constraints. Similar requirements occur in distributed control loops [52]. Thus far, the fact that communication usually comprises long sequences of tightly spaced packets has not been exploited to improve security.

Drone control [3] requires wireless transmission of thousands of small (a few bytes to tens of bytes in size) packets, fully utilizing the available bandwidth of the wireless channel [22]. Given their control of physical hardware, all commands must be authenticated individually in their specific context. As context and criticality of control messages varies over the course of operation, the required security level changes accordingly. In contrast, LoRaWAN devices in the IoT domain are optimized to transmit as few bytes, as rarely as possible due to energy limitations [2]. All unnecessary data and re-transmissions cause battery drain and diminish the overall utility and longevity of the system; in particular, this means that implicit resynchronization in cases of packet loss is preferable to explicit retransmissions. Even the cost of memory and the resulting storage limitations can limit the applicability of traditional MACs, as in the case of the *Memory Encryption Engine* in Intel SGX [25]. Various other domains share these requirements, such as in-car communication [50], haptic feedback controls [34], radio networks signalling [40], and even System/Network-on-Chip communication [39].

In summary, there are many practical use-cases that require authentication of message streams under the following conditions:

(1) Direct authentication of each packet immediately upon (or at most shortly after) its reception
(2) Tag sizes should be minimized to reduce bandwidth overhead
(3) High security guarantees, i.e. comparable to traditional MAC schemes
(4) Ability to resynchronize without explicit protocol communication

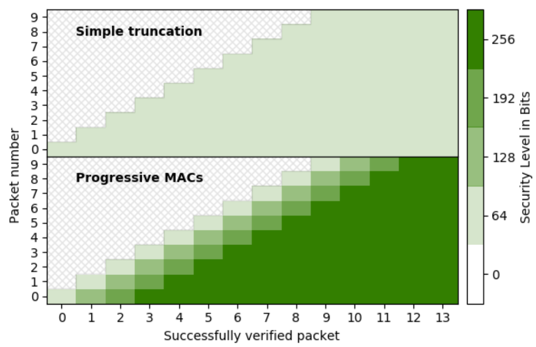


Figure 1: Achieved security level of progressive MACs (bottom), compared to simple truncation (top), with identical communication cost – tags truncated to 64 bits, internal MAC/key of 256-bit length each.

As discussed in Section 2 below, no prior work satisfies all of these conditions. To fill the gap, we propose the concept of Progressive MACs (ProMACs). In a nutshell, ProMACs extend (possibly truncated) MACs with a modest amount of dynamic internal state on the receiver side. This allows for implicit verification of older packets as more recent tags are received and results in progressively increasing level of security. As shown at the bottom of Fig. 1, each packet reception yields immediate integrity verification with the security level of 64 bits, which improves upon reception of subsequent packets. Individual tag sizes can still be chosen to be small enough to reduce communication and verification costs, or large enough for security critical packets. Moreover, internal state computation allows implicit resynchronisation without extra communication overhead.

The contributions of this paper are:

Formal Framework: We present a formal definition of ProMACs which formalizes stream integrity schemes that achieve increased security through progressive verification of packet streams. Moreover, we show that ProMACs extends the notion of standard MACs, i.e., any MAC scheme can be seen as an instantiation of ProMACs.

Generic Construction: We describe a generic construction of ProMACs based on pseudorandom functions (PRFs) and prove its security.

Experiments: Prototype implementations demonstrate the substantial speed-up and cost reduction in realistic settings.

The rest of the paper is organized as follows: Section 2 motivates aforementioned requirements and argues that prior schemes are unable to satisfy all of them. Next, Section 3 describes some necessary background concepts. Then, Section 4 introduces ProMACs in an intuitive manner, followed by a formal definition and security arguments, and concludes with the discsussion of ProMACs' relationship to classical MACs and duplex constructions. Section 5 constructs a ProMAC instance and proves its security properties. In Section 6, this construction is evaluated in realistic settings and its performance and security claims are evaluated. Section 7 concludes the paper.

2 REQUIREMENTS AND RELATED WORK

We start by motivating the requirements sketched out in Section 1. We then discuss relevant prior work and show that none of it satisfies all of these requirements, thus motivating a new design.

2.1 Requirements

We consider a scenario that involves two parties that share a secret k and exchange packet streams. Packets are sent over an insecure channel controlled by an active attacker that can inject, delete and manipulate traffic. Our objective is to protect the integrity of these streams at the packet level. Since the number of packets in a stream may not be known beforehand, or because different packets may have different levels of security criticality (see below), we have

Requirement I: Each packet should be authenticated immediately upon (or briefly after) its reception.

The common approach to packet stream integrity is to introduce a per packet MAC tag. However for high-frequency streams of short packets, transmission of a full MAC tag amounts to appreciable overhead, as can be seen from the ratio of payload- to tag-length. Applications controlling robots or drones, generate thousands of packets per second [34], and that their delivery has to be guaranteed with latency on the order of milliseconds [22], at an effective packet loss rate of under 10^{-9} in manufacturing environments. This agrees with [52] for Tactile Internet applications (or any control loop with haptic feedback) and [3] for drone control. For the latter, we expect packet sizes of dozens of bytes [3], while robot control packets are in a 15-20 byte range [19, 48]. Providing acceptable integrity of such a packet stream with a standard HMAC-SHA256 translates to extending each payload by 32 bytes. This results in an overhead of $\approx 200\%$, clearly rendering timely delivery for robot control applications impossible.

Another example is 802.15.4 – the foundation of ZigBee and one of the standard communication technologies for remote control. In 802.15.4, a millisecond delay can only be achieved with packet sizes under 30 bytes [24]. Consequently, we deduce

Requirement II: Tag sizes should be minimized to reduce communication overhead.

Of course it is possible to lower bandwidth overhead caused by MAC tags by simply reducing their bit-size. This yields a linear performance gain, as fewer bits are transmitted, yet it also results in an exponential security loss, since only transmitted tag bits are available to check each packet's integrity. This can threaten safety and security of underlying applications (see top of Fig. 1) and is generally not acceptable in all use-cases [41].

Approach	Req. I	Req. II	Req. III	Req. IV
Common MAC	✓	-	✓	✓
Truncated MAC	✓	✓	-	✓
MAC over aggregated packets	-	✓	✓	✓
Aggregated MACs	✓	✓	✓	-
Stateful MACs	✓	✓	✓	-
ProMAC	**✓**	**✓**	**✓**	**✓**

Table 1: Overview of other prominent MAC schemes with respect to requirements I–IV.

However, in several settings corruption of single packets in a stream might be acceptable, as long as most packets are verified *and* corruptions are still detected. This is particularly relevant to use-cases where retransmissions are undesirable, e.g., in video streaming, due to high data volume, a single corrupted frame might only be perceived as glitch, or in LoRaWAN sensors each packet transmission is costly and retransmissions must be avoided. Thus, detection of a single corrupted packet should be viewed as a normal event, which should not adversely affect the entire packet stream. Whereas, in prior techniques, a single packet authentication error leads to a stream reset, which includes tearing down the current session and establishing a new one. Consequently, it is desirable to have higher than the security of plain tag truncation.

Requirement III: High security guarantees in the long term, i.e. comparable to traditional MAC schemes and above the security of plain tag truncation.

Finally, communication (especially wireless) is subject to occasional transmission failures, e.g., malformed or dropped packets. Within the described use-cases it is much more sensible to keep the packet stream running and to allow for implicit resynchronization. Consequently, any practical security approach should support

Requirement IV: Ability to resynchronize without explicit protocol communication.

2.2 Related Work

We now demonstrate that prior relevant prior techniques do not satisfy all four requirements identified above. An overview is shown in Table 1.

2.2.1 Common MACs. As pointed out, the standard approach for ensuring packet integrity is to use common MACs, e.g., [4, 7, 10, 11, 27]; see Section 3.1 for a formal MAC definition. Althought appending a single full-blown MAC tag to each packet allows for the packet's immediate authentication upon receipt, the goodput is reduced by the size of the tag, which violates requirement II.

2.2.2 Truncated MACs. In settings that require low latency, and have constraints on packet size or energy consumption, regular MACs incur excessive overhead. To remedy the situation, it may be sufficient to protect packets from modification for a short time. Assuming that an adversary could neither predict nor forge a packet, and would have to find a packet along with a corresponding valid tag within a short period of time, some techniques offer lower security guarantees commensurate with reduced overhead.

This idea has been suggested for several specific cases. To achieve real-time communication on the CAN bus, [50] propose to add short MACs to CAN frames, evaluating performance for tags of various lengths. IEEE 802.15.4 (the foundation of ZigBee) also describes MAC truncation to as lowe as 32 bits [24], mainly to conserve transmission power and time. IPSec and DTLS are network- and session-layer protocols, respectively, that support truncated MACs of 96 [21, 51] and 80 [38] bits. Even the main signature and MAC standards, such as HMAC [35–37, 43], DSS [42], ISO/IEC 9797 [53, 54], SHA3-based cSHAKE, KMAC, TupleHash, and ParallelHash [31] support truncated MACs. However, using truncated MACs results in an exponential security loss in the number of truncated bits, hence violating our requirement III.

2.2.3 Aggregating Packets. A straightforward approach to satisfy requirements II and III (and, to some extent, IV) is to aggregate packets before generating a MAC, i.e., compute a tag over multiple packets. This way, instead of each packet carrying a MAC, only one out of every number of packets carries it. On prominent aggregation technique in [23] shows how each packet is transmitted with a hash of the subsequent packet, thus providing a means to detect modifications of future packets. The signature of the first packet includes the hash of the second packet and this signature is sent at the beginning of the stream. This approach requires advance knowledge of the entire packet stream. Another technique in [23] is based on online stream authentication, where trust is chained by sending the public key for verification of each subsequent packet. EMSS [44] and its several adaptations [14, 15] reverse the above idea by sending a full-length hash of previous packets to allow for modification detection without source authentication, and a full signature for authentication at the end of the stream. However, this easily allows an adversary to forge packets until a tag is actually required, which conflicts with requirement I.

2.2.4 Aggregated MACs. The idea behind aggregated MACs is that tags are computed individually for each packet and are then combined into a single tag. One example is XOR MAC [6] which does not provide a security level above truncated MACs, hence contradicting requirement III. Moreover, resynchronization is not directly possible (requirement IV). The concept of aggregated signatures was extended to MACs in [29] and was later generalized in [33] (see also [18]). While higher security levels are possible, no current scheme allows for direct resynchronization. To reduce MAC bandwidth overhead, Mini-MAC [49] adapts the resulting tag length according to the available space within its use-case. It respects stream characteristics by including a history of previous packets as MAC input. Besides the fact that including a packet history significantly raises the computation overhead, resynchronization is not supported.

2.2.5 Stateful MACs. Similar to our motivation, stateful MAC schemes focus on authenticity of data streams. A stateful MAC is similar to a classical MAC combined with an additional state at sender or receiver side. In this context, various security definitions and stream properties have been defined, e.g., [12, 13, 20, 28, 32, 46].

Current sponge-based constructions (e.g., Keccak [9], Blinker [47], Strobe [26] and Xoodoo [17]) can be used as stateful MACs to take advantage of the specific proprerties of packet streams. By operating in *MAC-and-continue* mode [47], internal state forwarding can be realized. Furthermore, the duplex operation mode [8] of sponge constructions can be used to output shorter tags to achieve statefulness and efficient transmission. The major distinction is that these constructions are fixed in their chaining approaches, i.e., using the whole preceding packet stream as a chain. This inherently prevents resynchronization of internal states in case of a disruption, thus violating requirement IV.

3 PRELIMINARIES

3.1 Message Authentication Codes

We use the standard definition of Message Authentication Codes (MAC) from [30].

Definition 3.1 (Message Authentication Codes (MACs)). A MAC scheme includes several sets and probabilistic algorithms. The sets are:

- $\mathcal{M}$ - the message space
- $\mathcal{K}$ - the key space
- $\mathcal{T}$ - the tag space

Furthermore, the following algorithms are involved:

(1) The *key-generation algorithm* Gen samples a secret key $k \in \mathcal{K}$
(2) The *tag-generation algorithm* Sig takes as input a key k and a message m and outputs a MAC tag t.
(3) The *verification algorithm* Vrfy takes as inputs a key k, a message m, and a tag t and outputs a decision $\delta \in \{\mathsf{true}, \mathsf{false}\}$. true indicates that the tag is valid, while false means that the tag is invalid.

3.2 Pseudorandom Functions

Our construction will be based on a pseudorandom function (PRF), as defined below.

Definition 3.2 (Pseudorandom Function). A function $F : \{0,1\}^\ell \times \{0,1\}^n \to \{0,1\}^m$ is called (q,t,ε)-pseudorandom if the following holds. Consider an algorithm $\mathcal{D}$, called the distinguisher. $\mathcal{D}$ gets black-box access to either F_k for a uniformly sampled $k \in \{0,1\}^\ell$ (with $F_k(\cdot) = F(k,\cdot)$), denoted by $\mathcal{D}^{F_k}$, or black-box access to a uniformly sampled function $f : \{0,1\}^n \to \{0,1\}^m$, denoted by $\mathcal{D}^f$. Eventually, $\mathcal{D}$ outputs a bit $b \in \{0,1\}$, denoted by $b \leftarrow \mathcal{D}$. Then, if $\mathcal{D}$ runs in time at most t and makes at most q queries to the black box, it holds that

$$\left| Pr\left[1 \leftarrow \mathcal{D}^{F_k}\right] - Pr\left[1 \leftarrow \mathcal{D}^{f}\right] \right| < \varepsilon. \quad (1)$$

4 PROGRESSIVE MACS

We start by overviewing our constructions, before formally defining ProMACs and their security properties.

4.1 Overview

Progressive MACs combine the concepts of aggregated packets and stateful MACs, mentioned in Section 2. At the beginning, both sender and receiver agree on the same internal, secret state, based on a shared secret key. For each packet, a tag is computed and sent together with the packet. The value of the tag depends on both the packet and the internal state. Moreover, the state is updated according to the current packet.

On the one hand, this allows immediate verification of packets (Requirement I). On the other hand, each tag now depends on both the current and previous packets. This yields progressive authentication, whereby future tags implicitly increase confidence of the integrity of earlier packets. Note that this allows us to use smaller tags (Requirement II) while a high level of security, with respect to tag integrity, is gradually achieved (Requirement III).

To support resynchronization, internal state update can be configured such that it depends only on a limited number of previous packets, denoted as *Area of Dependency*. Thus, even if a transmission error occurs, it is guaranteed that after correctly receiving a sufficient number of packets, the sender and receiver states are automatically synchronous again (Requirement IV). We want to point out that if fast resynchronization is necessary, one can choose an Area of Dependency of a rather short length. Even if the state depends only on the current and the previous packets, i.e., a length of two, it allows us to halve the length of tags and significantly reduce bandwidth overhead.

Fig. 2 depicts the generic ProMACs workflow.

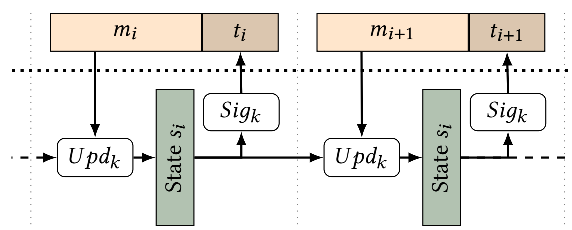


Figure 2: Generic workflow of a progressive MAC scheme. The tag generation Sig_k might produce short tags.

Security of ProMACs is intuitively determined by several parameters: Forging a single packet is as difficult as guessing a single truncated tag (either guessing the secret key, or the tag directly), yet only if no subsequent packets with their corresponding tags are received. Hence, an adversary has to prevent delivery of any tag subsequent to the forged packet, or actively guess all subsequent tags successfully, since the recipient would detect forgery otherwise.

Consequently, in *theory*, ProMACs security against existential forgery is the same as that of classical MACs: an attacker who aims to manipulate only the last packet of a ProMACs-protected packet stream must put in the same effort as in the case of a classical (truncated) MAC.

In *practice*, however, using ProMACs that incorporates information about previous packets in its internal state makes selective forgery more difficult in the following sense: the more packets are following in the stream after tampering, the more adversarial effort it takes, since the subsequent tags need to be forged as well. Note that if these tags are not forged accordingly, the attack is detected. That is, only in the case that an attacker aims to forge the last packet[1], security of a ProMACs falls back to the security of classical (possibly truncated MACs). In all other cases, ProMACs provides a higher level of security. Moreover, ProMACs can easily be extended to use varying-length tags. By using a higher tag length for the last packets, security of these would be at least as high as using a MAC with a standard security level, e.g., 128 bits.

[1]Note that it may not always be clear when the end of a packet stream is reached.

4.2 Formal Definition

We first provide a formal definition of ProMACs. This definition extends the notion of classical MACs [30]. While the extension is rather intuitive, defining the corresponding security properties is more subtle. The challenge is to correctly capture the kind of information an attacker may collect before attempting forgery. We provide a concise definition in Section 4.3. Note that our definition of ProMACs covers classical MACs as a special case. We discuss this in detail at the end of this section.

Definition 4.1 (Progressive MACs). A *progressive MAC* (ProMAC) scheme includes several sets and algorithms. The sets are:

- $\mathcal{M}$ - the packet space
- $\mathcal{K}$ - the key space
- $\mathcal{S}$ - the state space
- $\mathcal{T}$ - the tag space

Also, the following algorithms are included: (Gen, Init, Upd, Sig, Vrfy). We assume that the party that uses a ProMAC maintains an internal state from $\mathcal{S}$. The working principles of the algorithms are as follows:

(1) The probabilistic *key-generation algorithm* Gen samples a secret key $k \in \mathcal{K}$
(2) The probabilistic *initialization algorithm* Init samples an initial state $s_0 \in \mathcal{S}$
(3) The deterministic *update algorithm* $\mathsf{Upd} : \mathcal{K} \times \mathcal{S} \times \mathcal{M} \to \mathcal{S}$ takes as input a key $k \in \mathcal{K}$, a state $s \in \mathcal{S}$, and a packet $m \in \mathcal{M}$ and outputs a new state s'. We write this as $s' := \mathsf{Upd}_k(s, m)$
(4) The deterministic *tag-generation algorithm* Sig takes as input a key k, a state s, and a packet m and outputs a tag t. We write this as $t := \mathsf{Sig}_k(s, m)$.
(5) The deterministic *verification algorithm* Vrfy takes as inputs a key k, a state s, a packet m, and a tag t and outputs a decision $\delta \in \{\mathsf{true}, \mathsf{false}\}$. The output true indicates that the tag has been accepted while it has been rejected in the case of false. We write this as $\delta := \mathsf{Vrfy}_k(s, m, t)$.

We chose our algorithms Sig and Upd to be deterministic, since in the case of probabilistic algorithms we would have to transmit additional information, which conflicts with our primary objective, to achieve efficient communication.

While classical MACs operate independently on *single packets*, ProMACs are meant to be used for integrity of *packet streams*. To this end, the workflow is as follows (see Fig. 2): Initially, a secret key is generated by executing Gen. Then, for each packet stream a random initial state s_0 is picked by Init. The random initial state is communicated in the clear to other parties. Given this, for each packet m_i in the packet stream, the state is updated using Upd and the next tag is produced via Sig from this state. This results into a stream of the form $s_0, m_1, t_1, m_2, t_2, \ldots$. Verifiers use s_0 as initial state and then subsequently update the state using Upd and the incoming packets and finally validate the corresponding tags using Vrfy.

4.3 Correctness and Soundness

4.3.1 Correctness. Correctness means that, for every key k output by Gen, for every initial value $s_0 \in \mathcal{S}$, for every $j \geq 1$, and for every sequence of packets $(m_1, \ldots, m_j) \in \mathcal{M}^j$, it holds that if the corresponding tags $(t_1, \ldots, t_j)$ are honestly computed as outlined, the verification algorithm Vrfy accepts all of these.

Defining the notion of security is less straightforward for the the following two reasons. First, the common security model for MACs allows an attacker to make sign queries to an oracle to learn the tags for selected packets, i.e., the input to the Sig algorithm. However, our situation is different since the tag is computed from an internal state unknown to an attacker.

Second, for practical reasons we want to support easy resynchronization, i.e., even if some packets in a stream are lost, it should be possible to validate the remaining packets. Technically, this means that both sender and receiver should eventually reach the same state again, even if some packets are lost. To this end, we introduce an additional notion, Area of Dependency, with respect to the update function that reflects how many successive packets determine the next state.

4.3.2 Area of Dependency. Before we define Area of Dependency, we need to extend our notation. We write $\mathsf{Upd}_k^i(s_0, m_1, \ldots, m_i) = s_i$ to reflect that applying i-times the update function on initial state s_0 with key k on packets $m_1, \ldots, m_i$ results into state s_i. That is, we have:

$$\mathsf{Upd}_k^1(s_0, m_1) = \mathsf{Upd}_k(s_0, m_1)$$
$$\mathsf{Upd}_k^2(s_0, m_1, m_2) = \mathsf{Upd}_k(\mathsf{Upd}_k(s_1, m_1), m_2)$$

and so on.

Definition 4.2 (Area of Dependency). Consider a ProMAC with an update function Upd. We say that Upd has $(u + 1)$-independence if the following holds: for any state $s \in \mathcal{S}$, any key k given by Gen, and any packets $m_2, \ldots, m_{u+1}$, there exists a state s', such that

$$\mathsf{Upd}_k^{u+1}(s, m', m_2, \ldots, m_{u+1}) = s' \quad \forall m' \in \mathcal{M}. \tag{2}$$

The Area of Dependency of Upd, denoted by $\mathsf{ad}(\mathsf{Upd})$, is defined to be smallest u such that Upd has $u + 1$-independence. If all previous packets might impact the current state, we write $\mathsf{ad}(\mathsf{Upd}) = \infty$.

Note that this definition requires that the condition must hold for *any* state s and is not restricted to states sampled by Init. The intent of this definition is that an update function Upd with $\mathsf{ad}(\mathsf{Upd}) = u$ has the property that the current state is independent from the $(u + 1)^{\text{th}}$-last packet. Consequently, each state depends on (at most) last u packets, the initial state, and the current index, as we show next:

Proposition 4.3. *Consider a ProMAC with an update function* Upd *with* $\mathsf{ad}(\mathsf{Upd}) = u$*. Let* $i > u$ *and*

$$\mathsf{Upd}_k^i(s_0, m_1, m_2, \ldots, m_i) = s_i. \tag{3}$$

for some arbitrary initial state s_0 *given by* Init*, arbitrary key* k *given by* Gen*, and arbitrary packets* $m_1, \ldots, m_i$*. Then, it holds that* s_i *and all follow-up states* $s_{i+1}, s_{i+2}, \ldots$ *are independent from the content of the packets* $m_1, \ldots, m_{i-u}$*. In other words, the current state* s_i *can depend only on the initial state, the key, the last* $u - 1$ *packets and the index* i*. That is, the current state is independent of the last-but-u packets, and depends only on the initial state and the last* $u - 1$ *packets (or less).*

Proof. We show the claim by induction over i.

Let $i := u+1$. Following Definition 4.2, s_i is independent from m_1. Now let $j \geq 1$ be arbitrary. Since $s_{i+j} = \mathsf{Upd}_k^j(s_i, m_{i+1}, \ldots, m_{i+j})$, this state is independent from from m_1 as well.

Now assume that the claim holds for some $i \geq u+1$, i.e., all states $s_i, s_{i+1}, \ldots$ are independent from packets: $m_1, \ldots, m_{i-u}$. It suffices to show that states: $s_{i+1}, \ldots$ are independent from m_{i-u+1}. The fact that s_{i+1} is independent from m_{i-u+1} follows from Definition 4.2 (note that the definition is not restricted to states sampled by Init) and the fact that $s_{i+2}, \ldots$ are independent from m_{i-u+1} can be shown as above for $i = u + 1$. □

A consequence of Proposition 4.3 is that for each $i \geq 1$, there exists a procedure $\mathsf{Upd}^{[i]}$, such that for all initial states s_0 given by Init, all keys k given by Gen, and all packets $m_1, \ldots, m_i$, it holds that:

$$\mathsf{Upd}_k^i(s_0, m_1, \ldots, m_i) = \mathsf{Upd}_k^{[i]}(s_0, i, m_{\max\{1, i-u+1\}}, \ldots, m_i). \quad (4)$$

That is, $\mathsf{Upd}^{[i]}$ gets as input only the initial state s_0, the index i, and the last $u - 1$ packets (or less if $i < u - 1$), but computes the same internal state as Upd^i.

4.3.3 Soundness. We now define security of ProMAC based on a game played by an attacker $\mathcal{A}$ and an oracle O. Although the security definition is similar to that of traditional MACs, it needs to take into account the concepts of state and Area of Dependency to correctly cover the notion of forgery.

The game works as follows. At the beginning, O samples a secret key k and initializes an empty set $Q = \emptyset$. Also, let $\mathsf{ad}(\mathsf{Upd}) = u < \infty$.[2]

The attacker can now make Sig-queries to O. Formally, it sends a sequence of initial state, an index i, and packets and receives the corresponding tag of the last packet. To this end, we distinguish between $i \leq u$ and $i > u$.

- $i \leq u$: The attacker sends a query $Q := [s, i, m_1, \ldots, m_i] \in \mathcal{S} \times \mathbb{N} \times \mathcal{M}^i$ to O where $\mathbb{N}$ denotes the set of positive integers and s is some initial state. The oracle O first computes $s' := \mathsf{Upd}_k^{i-1}(s, m_1, \ldots, m_{i-1})$ where we set $s' := s$ in case of $i = 1$. Afterwards, it computes $t := \mathsf{Sig}_k(s', m_i)$, appends it to Q, and returns t to $\mathcal{A}$.
- $i > u$: The attacker sends a query $Q := [s, i, m_1, \ldots, m_u]$ to O. The oracle determines $s' := \mathsf{Upd}_k^{[u-1]}(s, m_1, \ldots, m_{u-1})$ and then $t := \mathsf{Sig}_k(s', m_u)$. Finally, it appends t to the query Q and returns t to the attacker.

In both cases, the sequence Q is inserted into $\mathcal{Q}$, i.e., $\mathcal{Q} := \mathcal{Q} \cup \{Q\}$. We call the elements stored in $\mathcal{Q}$ the *query sequences*.

Eventually, the attacker $\mathcal{A}$ outputs potential forgery, in the form of a sequence: $Q^* := [s_0^*, m_1^*, t_1^*, \ldots, m_r^*, t_r^*]$ for $r \geq 1$. The attacker *wins* the game if the tags are accepted while not all of them have been previously queried *in this form* by the attacker. To make "in this form" more precise, we need to take a look at all subsequences from Q^*. These are defined as follows, where $u := \mathsf{ad}(\mathsf{Upd})$:

$$\begin{aligned}\mathcal{Q}^* := \{&[s_0^*, 1, m_1^*, t_1^*], \ldots, [s_0^*, u-1, m_1^*, \ldots, m_u^*, t_u^*],\\ &[s_0^*, u+1, m_2^*, \ldots, m_{u+1}^*, t_{u+1}^*], \ldots,\\ &[s_0^*, r, m_{r-u+1}^*, \ldots, m_r^*, t_r^*]\}.\end{aligned}$$

[2]The case of $\mathsf{ad}(\mathsf{Upd}) = \infty$ will be discussed at the end.

We call the elements of $\mathcal{Q}^*$ the *forgery sequences*. Note that for each $\mathcal{Q}^*$ that also appears in $\mathcal{Q}$, it holds that the attacker already knew that this combination of initial state, index, and packets leads to the respective tag. Thus, the attacker wins if at least one of these forgery sequence has not been asked before, i.e., is not equal to a query sequence stored in $\mathcal{Q}$. We denote forgery sequences that are not part of $\mathcal{Q}$ as *fresh forgery sequence*. Thus, the winning condition can be reformulated to mean that an attacker produces a packet stream with associated tags, such that: (i) all tags are accepted, and (ii) at least one forgery sequence is fresh, i.e., $\mathcal{Q}^* \not\subseteq \mathcal{Q}$.

We define a progressive MAC to be (q, Δ, ϵ)-secure if no adversary that makes at most q Sig-queries can succeed in the above experiment to generate Δ fresh forgery sequences with probability above ϵ, i.e.,

$$Pr\left[\mathcal{A} \text{ wins}\right] \leq \varepsilon. \quad (5)$$

We say that the scheme is (q, ϵ)-secure if the attacker can freely choose the number $\Delta \geq 1$ of fresh sequences.

The Case of $\mathsf{ad}(\mathsf{Upd}) = \infty$. The treatment of the case $\mathsf{ad}(\mathsf{Upd}) = \infty$ is quite similar. In a nutshell, the difference is that the queries contain all i packets and not the last u packets only. For instance, the forgery sequences are $[s_0^*, i, m_1^*, \ldots, m_i^*, t_i^*]$ for $i = 1, \ldots, r$.

4.4 Relation to Classical MACs and Duplex-based Constructions

Before discussing concrete ProMACs instantiations of ProMACs in the next section, we note that the definition (including the notion of security) extends the concept of classical MACs that operates on $\mathcal{M}$ and also stateful MACs as duplex-based constructions.

More precisely, let $\widetilde{M}$ denote a classical MAC with algorithms $\widetilde{M}$.Gen, $\widetilde{M}$.Sig, and $\widetilde{M}$.Vrfy. We define a ProMAC M based on $\widetilde{M}$ by essentially ignoring the internal state. That is we set $\mathcal{S} = \{s_0\}$ for some dummy value s_0 and define $\mathsf{Upd}_k(s_0, m) := s_0$ for any packet m. Finally, we set $M.\mathsf{Gen} := \widetilde{M}.\mathsf{Gen}$, $M.\mathsf{Sig}_k(s_0, m) := \widetilde{M}.\mathsf{Sig}_k(m)$, and $M.\mathsf{Vrfy}_k(s_0, m, t) := \widetilde{M}.\mathsf{Vrfy}_k(m, t)$ for all keys k and all packets m. Note that $\mathsf{ad}(\mathsf{Upd}) = 1$, i.e., current state depends only on the current packet. Consequently, all query sequences stored during the security game have the form $[s_0, m, t]$ where s_0 can be ignored. Thus, both the working principle and security of M is effectively equivalent to $\widetilde{M}$.

On the other hand, if we set $\mathsf{ad}(\mathsf{Upd}) = \infty$, i.e., each tag depends on *all* packets so far, we have a stateful MAC similar to the duplex-based constructions. In that sense, ProMACs can be seen as a tradeoff between both constructions.

Finally, we stress that a ProMAC can be expressed as a classical, deterministic MAC with packets of the form: $[s, i, m_1, \ldots, m_{\min\{u,i\}}]$ (see Definition of Sig-queries in Section 4.3.3). In particular, for each such query the resulting tag there exists exactly one tag. Using the notions and arguments from [5] (which also considers MACs that operate on packets of varying lengths), it follows that: (i) the security definition given in Section 4.3 captures SUF-1 (strong unforgeability with single access to a verification algorithm), and (ii) SUF-1 implies SUF-m (strong unforgeability with multiple access to a verification algorithm). Therefore, it is unnecessary to define verification queries in the security game.

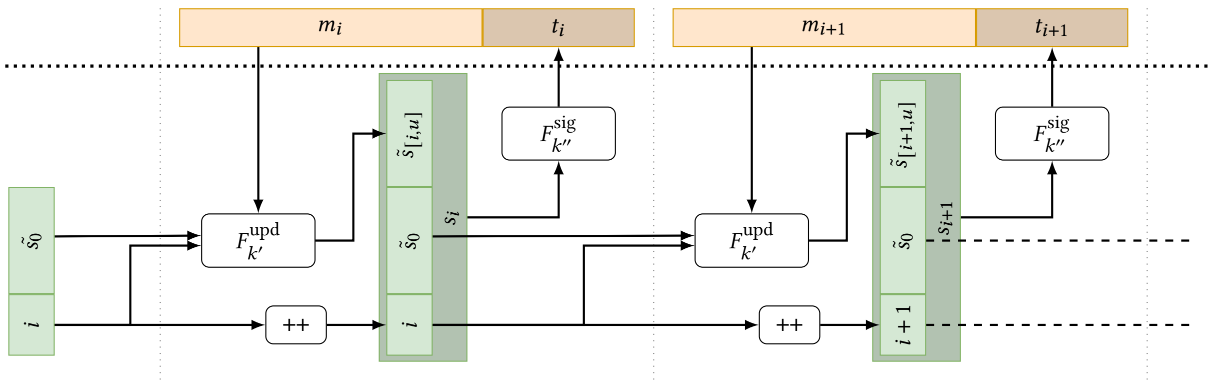


Figure 3: Core concept of the ProMAC instantiation *Whips*, with state update Upd **and tag generation** Sig **realised through one PRF each.**

5 WHIPS

In this section, we present a concrete ProMACs instantiation, dubbed Whips [3]. It uses two pseudorandom functions (PRFs) $F^{\text{upd}}_{k'}$ and $F^{\text{sig}}_{k''}$ as building blocks and allows one to freely choose the tag size. This makes them compelling candidates for wireless communication. To achieve high bandwidth efficiency, the practical constructions employ a small tag space, i.e., the tag length τ is small (which resembles simple truncation). Moreover, it allows re-synchronization after u packets.[4] To this end, the current state is composed of a counter and u so-called substates where one packet determines one substate.

5.1 Specification

We start by defining the sets:

- $\mathcal{M} := \{0,1\}^{\mu}$ - the packet space
- $\mathcal{K} := \{0,1\}^{\kappa^{\text{upd}}+\kappa^{\text{sig}}}$ - the key space
- $\mathcal{S} := \{0,1\}^{\gamma+(u+1)\cdot\sigma}$ - the state space with γ being the length of a counter and σ being the length of *substates*
- $\mathcal{T} := \{0,1\}^{\tau}$ - the tag space

Whips maintains internal state s_i composed of: counter i, initial substate $\tilde{s}_0 \in \{0,1\}^{\sigma}$, and u previous substates $\tilde{s}_{i-u}, \ldots, \tilde{s}_{i-u} \in \{0,1\}^{\sigma}$. To simplify the description, we define the term $\tilde{s}_{[i,u]} \in \{0,1\}^{u\cdot\sigma}$ for $i \geq 0$ as:

$$\tilde{s}_{[i,u]} := \begin{cases} \tilde{s}_{i-u+1}, \tilde{s}_{i-u+1}, \ldots, \tilde{s}_i & , i-u+1 \geq 0 \\ \underbrace{\tilde{s}_0, \ldots, \tilde{s}_0}_{u-(i+1)\ \text{times}}, \tilde{s}_0, \tilde{s}_1, \ldots, \tilde{s}_i & , \text{else} \end{cases} . \tag{6}$$

Using this, internal state is defined as:

$$s_i = (i, \tilde{s}_0, \tilde{s}_{[i,u]}), \tag{7}$$

where s_0 is the initial state. In each round, the next packet m_{i+1} is processed to compute: (i) the next state s_{i+1}, and (ii) the next tag t_{i+1}. To this end, the two PRFs: $F^{\text{upd}}_{k'}$ and $F^{\text{sig}}_{k''}$. are used, respectively. Below we assume that the sampled key k has the form $k = (k', k'')$ and that these two parts have been used to initialize the two PRFs. $F^{\text{upd}}_{k'}$ takes as input the counter, the initial substate, and the packet and outputs the next substate:

$$F^{\text{upd}}_{k'} : \{0,1\}^{\gamma} \times \{0,1\}^{\sigma} \times \{0,1\}^{\mu} \;\rightarrow\; \{0,1\}^{\sigma} \tag{8}$$

$$(i, \tilde{s}_0, m_{i+1}) \;\mapsto\; \tilde{s}_{i+1} \tag{9}$$

This defines the next state $s_{i+1} = (i+1, \tilde{s}_0, \tilde{s}_{[i+1,u]})$ which is used to compute the corresponding tag with the help of $F^{\text{sig}}_{k''}$:

$$F^{\text{sig}}_{k''} : \{0,1\}^{\gamma} \times \{0,1\}^{\sigma} \times \{0,1\}^{u\cdot\sigma} \;\rightarrow\; \{0,1\}^{\tau} \tag{10}$$

$$(i+1, \tilde{s}_0, \tilde{s}_{[i,u]}) \;\mapsto\; t_i \tag{11}$$

The core idea of the proposed instantiation is shown in Fig.3. We now discuss the algorithms that are part of the model, per Definition 4.1:

Key Generation Gen. The probabilistic key-generation algorithm Gen samples a secret key $k \in \mathcal{K}$. Below we assume that sender and receiver share a common secret key $k \in \{0,1\}^{\kappa}$ and write $F_k(\ldots)$ instead of $F(k, \ldots)$.

Initialization Init. The probabilistic initialization algorithm Init samples an initial substate $\tilde{s}_0 \in \{0,1\}^{\sigma}$ and sets[5]

$$s_0 = (0, \tilde{s}_0, \tilde{s}_{[0,u]}). \tag{12}$$

Also, it samples a key $k = (k', k'') \in \mathcal{K}$ to initialize the PRFs $F^{\text{upd}}_{k'}$ and $F^{\text{sig}}_{k''}$.

Update Upd *and Tag Generation* Sig. Both procedures: state update and next tag computation, are accomplished by a call to one PRF. [6] More precisely, let s_{i-1} denote the current internal state (with s_0 being the initial state) and let m_i be the current packet. Then, Whips updates the internal state by computing:

$$\tilde{s}_i := F^{\text{upd}}_{k'}(i-1, \tilde{s}_0, m_i) \tag{13}$$

[3] The inspiration for the name is that the sliding "Area of Dependency" resembles the moving "wave" when cracking a whip.

[4] In the following, we consider the case where u is finite. At the end, we shortly discuss the case of $u = \infty$.

[5] The first $u-1$ states contain multiple copies of $\tilde{s}_0$ to be compatible with the overall formal of all states.

[6] In fact, one could replace the second PRF $F^{\text{sig}}_{k''}$ by a classical MAC. We chose to rely on one type of cryptographic primitive, since it may be more efficient in practice to implement it only once and use it with different keys, e.g., hash functions.

where $u = \mathrm{ad}(\mathrm{Upd})$ is the Area of Dependency. This defines $s_i = (i, \tilde{s}_0, \tilde{s}_{[i,u]})$. Given this, Whips computes the tag t_i for m_i by

$$t_i := F^{\mathrm{sig}}_{k''}(s_i). \quad (14)$$

Verification. The verification algorithm Vrfy_k computes on input $s_i, m_i, \tilde{t}_i$ first the tag t_i and then compares it to the given $\tilde{t}_i$. If both are equal, the output is true; otherwise it is false.

The Case of Infinite u. If $u = \infty$, i.e., the equivalent of duplex-based chaining, it is no longer necessary to store the last u substates to "cancel" these out later. Instead, more compact solutions are possible, e.g., setting $s_i = (i, \tilde{s}_0, \tilde{s}_i)$ and choosing an appropriate PRF F. Here, we can take advantage of the fact that $\tilde{s}_i$ anyhow depends on all previous states. For security reasons, it is necessary to choose a higher value for σ.

5.2 Design Rationale

Before proving security of our construction, we briefly discuss why s_i contains the counter, the initial state, and the recent substates.

Let us assume that the state did not contain a counter. Recall that one goal of our construction is to realize an Area of Dependency to support resynchronization. This means that, in a scenario where the packet stream consists of the repetition of the same packet, i.e., $m_0 = m_1 = \ldots$, the tags would all be the same, which would allow for a simple forgery attack. The counter ensures freshness of the state, even if the packets repeat.

Next, we assume that the state would not contain the initial state value s_0. Then, an attacker who can figure out the internal state s_i for $i > 0$ could produce a forgery by using s_i as "initial state". Hence, the initial state, which is beyond attacker's control, acts as anchor for the trust chain. In addition, it prevents replay attacks in the sense that tags observed for one initial state are re-used for a different initial state.

Finally, to facilitate removal of "outdated" substates from the current state calculation (to ensure an Area of Dependency of length u), the current state is mainly composed from most recent u substates.

5.3 Proof of Security

Below, we prove security of our construction. Recall that the motivation of ProMACs is an attacker who aims to forge a certain tag also has to forge upcoming tags. Consequently, the number of tags to be forged needs to be a part of the security claim. In the context of the security definition, this is equivalent to the number of fresh forgery sequences Δ (see the definition in Section 4.3.3). The security claim is as follows:

THEOREM 5.1. *Consider an instantiation of Whips with two $(q, T, \varepsilon/2)$-pseudorandom functions F^{upd} and F^{sig}, with substate length of σ, tag length of $\tau \geq 2$, and counter length of at least $\gamma \geq \log_2(q)$. Then, for any attacker $\mathcal{A}$ who runs in time at most T, makes at most q Sig-queries and produces Δ fresh forgery sequences, the probability of success is upper-bounded by:*

$$Pr\,[\mathcal{A}\text{ wins}] \leq \varepsilon + \frac{q^2 + 2q}{2^\sigma} + \left(\frac{1}{2^\tau}\right)^\Delta. \quad (15)$$

In the parameter choices, we suggest keeping τ small (since it impacts bandwidth overhead), while σ can be large (since states is only stored internally). Thus, $\frac{1}{2^\tau}$ can be seen as the dominating term of the sum.

We note that (15) also describes the increasing security level of the proposed scheme. Since the attacker's advantage decreases with every subsequent successful verification, i.e. with increasing Δ, guaranteed security level for the respective packet is increasing with each of these packets. For each verified packet, the security level increases by transmitted tag bits with an upper bound of the key length. Fig. 1 (bottom) depicts such an increasing security level, while Fig. 4 demonstrates resynchronization properties of *Whips* and the increasing security level.

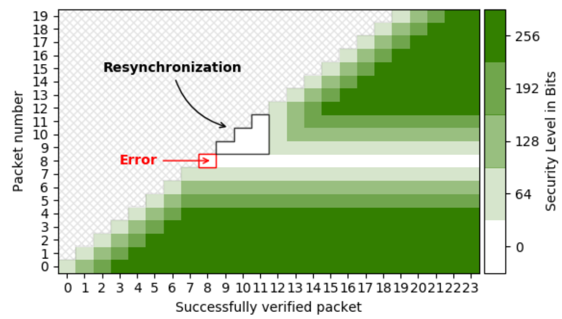


Figure 4: Achieved security level for the respective incoming packet, assuming a 64-bit tag size and 256-bit key size, $\mathrm{ad}(\mathrm{Upd}) = 4$ **and an error at packet #8.**

Finally, we note that the proof is independent of u, i.e., the given bound holds for both finite and infinite u.

PROOF OF THEOREM 5.1. We show the security claim (15) by using a sequence of games G_0-G_3. Since $\gamma \geq \log_2(q)$, we exclude the case of counter overflow, i.e., that the counter repeats after up to q queries.

Game G_0. Let G_0 denote the original security game as described in Section 4.3. We are interested in showing an upper bound for $Pr\,[\mathcal{A}$ wins in $G_0]$.

Game G_1. G_1 is defined as G_0 with the only difference that, whenever the attacker makes a Sig-query, it learns the full output of $F^{\mathrm{upd}}_{k'}$. Since the attacker now has more information, the probability of success is at least as high as before:

$$Pr\,[\mathcal{A}\text{ wins in }G_0] \leq Pr\,[\mathcal{A}\text{ wins in }G_1]\,. \quad (16)$$

Game G_2. G_2 is based on G_1 with the difference that the PRFs are replaced by a randomly chosen functions. Because each PRF is $(q, t, \varepsilon/2)$-pseudorandom and that $\mathcal{A}$ runs in time at most t and makes at most q queries, it holds that:

$$Pr\,[\mathcal{A}\text{ wins in }G_1] \leq Pr\,[\mathcal{A}\text{ wins in }G_2] + \varepsilon. \quad (17)$$

Game G_3. G_3 is defined as G_2 with the difference that the game is aborted if during the Sig-queries, a collision in the states occurs. An obvious upper bound is given by considering a collision on the next substate only. Since F_k is replaced by a random function, it follows that:

$$Pr\,[\mathcal{A}\text{ wins in }G_2] \leq Pr\,[\mathcal{A}\text{ wins in }G_3] + q^2/2^\sigma. \quad (18)$$

It remains to upper bound $Pr[\mathcal{A}$ wins in $G_3]$.

We assume that $\Delta = 1$. Then, an attacker wins if: (i) either it can produce a state collision or, (ii) a tag collision. That is, we have:

$$Pr[\mathcal{A} \text{ wins in } G_3 | \Delta = 1] = \frac{q}{2^\sigma} + \left(1 - \frac{q}{2^\sigma}\right) \cdot \frac{1}{2^\tau}. \quad (19)$$

Below we use $c_i := \frac{q+i}{2^\sigma}$ where c stands for "collision". We can rephrase the equation as:

$$Pr[\mathcal{A} \text{ wins in } G_3 | \Delta = 1] = c_0 + \frac{1 - c_0}{2^\tau}. \quad (20)$$

Next, we consider the case of $\Delta = 2$. For the first fresh sequence, there are two possibilities: (i) a state collision, or (ii) a tag collision. In the former, it follows that the second fresh sequence comes "for free". In the latter, we again have the same possibilities for the second fresh sequence. This yields:

$$Pr[\mathcal{A} \text{ wins in } G_3 | \Delta = 2] = c_0 + \frac{1 - c_0}{2^\tau} \cdot \left[c_1 + \frac{1 - c_1}{2^\tau}\right]. \quad (21)$$

By induction and using the fact that $1 - c_i \le 1$ for all $i \ge 0$, we can show that probability of winning G_3 with Δ fresh sequences is upper bounded by:

$$c_0 + \frac{c_1}{2^\tau} + \frac{c_2}{2^{2\tau}} + \ldots + \frac{c_{\Delta-1}}{2^{(\Delta-1)\cdot\tau}} + \frac{1}{2^{\Delta\cdot\tau}} = \sum_{i=0}^{\Delta-1} \frac{c_i}{2^{(i)\cdot\tau}} + \left(\frac{1}{2^\tau}\right)^\Delta. \quad (22)$$

Note that, for the sum, it holds that for increasing Δ, the number of terms in the sum increases, while their values decrease. Thus, we aim to find an upper bound for the sum. Let $i \ge 0$ be arbitrary. For the ratio of two successive, it holds that:

$$\begin{aligned}\left(\frac{c_{i+1}}{2^{(i+1)\cdot(\tau+1)}}\right) / \left(\frac{c_i}{2^{i\cdot\tau}}\right) &= \frac{q+i+1}{q+i} \cdot \frac{1}{2^\tau} = \left(1 + \frac{1}{q+i}\right) \cdot \frac{1}{2^\tau} \\ &\le 2 \cdot \frac{1}{2^\tau} \le \frac{1}{2}.\end{aligned}$$

In other words, each term in the sum is at most half the size of the previous term. This allows us to to derive the following upper bound:

$$\sum_{i=0}^{\Delta-1} \frac{c_i}{2^{i\cdot\tau}} \le c_0 + \frac{c_0}{2} + \frac{c_0}{2^2} + \ldots + \frac{c_0}{2^{\Delta-1}} < 2 \cdot c_0 = \frac{2q}{2^\sigma}. \quad (23)$$

Thus, it follows from equations (22) and (23), that attacker's probability of success in game G_3 with Δ fresh sequences can be upper-bounded by:

$$\frac{2q}{2^\sigma} + \left(\frac{1}{2^\tau}\right)^\Delta. \quad (24)$$

Putting inequalities (16), (17), (18), and (24) together, our claim follows. □

5.4 Parameter Selection

We assume the desired security level of λ against an attacker that makes up to q Sig-queries and aims for Δ fresh sequences. Next, we discuss how to choose Whips parameters to meet these requirements. To this end, the upper bound:

$$\varepsilon + \frac{q^2 + 2q}{2^\sigma} + \left(\frac{1}{2^\tau}\right)^\Delta \quad (25)$$

provided by Theorem 5.1 can guide parameter selection. Note that the theorem already assumes that the counter is chosen to be sufficiently large as to avoid any repetitions, i.e., $\gamma \ge \log_2(q)$. The values involved in this bound are:

- $\varepsilon/2$ – PRF security levels (which also depends on key lengths: κ^{upd} and κ^{sig})
- σ– substate size
- Δ– tag length

To simplify the following discussion, we investigate how to ensure that each of the three terms of the sum is below $2^{-\lambda}$. This results into a slightly higher upper bound of $3 \cdot 2^{-\lambda}$, instead of $2^{-\lambda}$. Alternatively, we could upper-bound each term by $2^{-\lambda-2}$.

Ensuring that $\varepsilon \le 2^{-\lambda}$ depends on the choice of PRFs. A necessary condition is to set the key lengths $\kappa^{\text{upd}}, \kappa^{\text{sig}} \ge \lambda$.

With respect to the second term, if we consider q^2 to be dominating, the following condition holds for the length of the substate:

$$\sigma \ge 2\log_2(q) + \lambda. \quad (26)$$

Since state size is $\gamma + (u+1) \cdot \sigma$, its lower bound is:

$$\log_2(q) + (u+1) \cdot (2\log_2(q) + \lambda). \quad (27)$$

Although this can be relatively large, it is acceptable in practice since storing data is far cheaper than sending it.

Finally, the third term implies the following inequality:

$$\tau \ge \lambda/\Delta. \quad (28)$$

With respect to the PRFs, $F^{\text{upd}}_{k'}$ and $F^{\text{sig}}_{k''}$ must yield output of sizes σ and τ, respectively. To this end, the bounds given in (26) and (28) can be used. We illustrate this with some concrete numbers. Assuming the goal of $\lambda = 128$ bit security, we allow $q = 2^{64}$ queries, and support $u = 4$. Then, state size has to be at least:

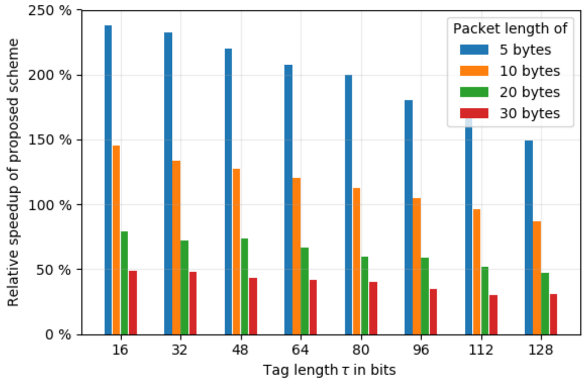
$$\log_2(q) + (u+1) \cdot (2\log_2(q) + \lambda) = 64 + 5 \cdot (128 + 128) = 1344 \quad (29)$$

bits or 168 bytes. F^{upd} must produce outputs of $2\log_2(q) + \lambda = 128 + 128 = 256$ bits, and F^{sig}– $\lambda/\Delta = 128/\Delta \ge 128$ bits. Thus, any modern hash function could be used for realizing these two PRFs.

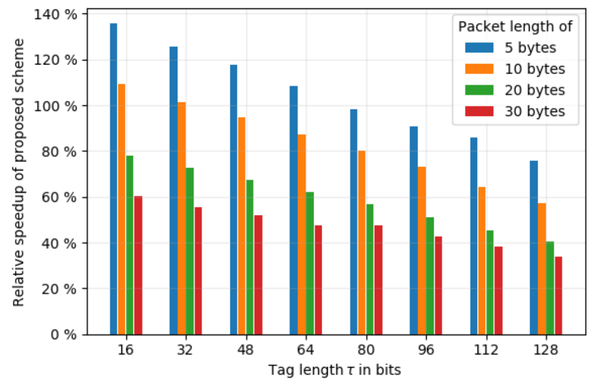
6 PERFORMANCE EVALUATION

In this section, we evaluate performance of the proposed *Whips* schemes in real-world scenarios. In doing so, we aim to understand the impacts of: computational overhead, energy savings, reduced latency and validation errors. We also evaluate throughput gains for various application and network settings.

Since this intends to be the main application, we first measure the performance of *Whips* as drop-in replacement for integrity schemes in robot control and WiFi communication, in experiments on realistic hardware. Considering the dependency on reliable delivery and focus on resynchronization capabilities as well as the fact that our realistic scenarios are characterized by comparatively low packet loss, we subsequently extend the study and analyze how performance and security evolve with increasing packet loss. Finally, we analyse the computation overhead, latency gains and energy savings of the proposed constructions. The results underline that the combination of truncation and state chaining leads to both transmission speedups *and* increased security levels in all scenarios with somewhat realistic communication settings.



(a) Experiments with TMote Sky using 802.15.4



(b) Experiments with WiFi 802.11a 5GHz

Figure 5: Relative speedup over traditional integrity schemes (HMAC-SHA256)

6.1 Evaluation Scenarios and Setups

We consider two typical settings, where streams of short messages are prevalent: (1) robot and drone control, and (2) general WiFi communication.

Robot control assumes a constant stream of very small messages (5 to 50 Bytes), transmitted over low power wireless networks, such as 802.15.4, at a transfer rate of 250 kbps [3]. The transmission latency is required to be around or under $1ms$, to prevent oscillations and reaction coupling [52]. We employ Tmote Sky motes for this experiment, which are popular and have been used in several prior studies [45].

WiFi communication Our results for WiFi communication are obtained from measurements with two laptops, using Intel Dual Band Wireless AC (2 x 2) 8265 and 8260 Chips.

The following **Experimental Setup** was used: All experiments were conducted using two devices of the respective scenario, in a typical office environment. We measured throughput, as well as error characteristics for both our proposed and the traditional integrity schemes under test, using especially crafted software on the Tmotes, and iperf3 on the laptops. We assume reliable communication at full integrity verification throughout all measurements: lost or malformed packets are recovered using selective repeat request. The setup of the robot control was chosen as described above, while for the WiFi setup we connected two laptops via WiFi AdHoc, on channel 129 of the 5GHz band. We started our experiments with 1-byte payload messages, and increased payload size by 1 Byte for each subsequent measurement. We compare the results of *Whips* to measurements using traditional HMAC-SHA256 as the baseline.

In this section we demonstrate the major benefit of the proposed schemes: combining the speedup of transmitting truncated tags with a high effective security level.

6.2 Empirical Performance Measurements

To assess performance gains we measured *Whips* in realistic scenarios.

We ran several experiments using the robot control and WiFi communication setup. As the length of the payload has the most pronounced impact on speedup, we focused on varying payload length.

Given the requirements for reliable communication with high security, we expect *Whips* to achieve appreciable speedup over traditional MACs. The essential advantage is that shorter tags are transmitted than with traditional MAC schemes achieving similar security. The relative speedup depends on actual reduction of transmitted bits per message, where message and headers remain identical. The main differences in the measurements will hence be caused by differing headers for the networking technologies, and the effect of message loss.

Figure 5 presents our results for selected packet sizes, which generally confirm our expectations. We observe a relative speedup of over 200% for *Whips* with tag length < 80 for small messages. The speedup drops to around 30-50% for message of 30 Bytes on motes (robot control), and around 35-60% on laptops. It slowly drops further with increasing message sizes. As WiFi is very stable, we did not measure any significant packet loss.

In summary, measurements in the described use case scenarios showed, that the proposed *Whips* scheme yield significant performance increases, especially for the message sizes we expect to see in robot and drone control.

6.3 Achieved Security Level

Whips is expected to provide considerably stronger security guarantees than truncated MACs, as shown in Eq. (15) and Fig. 4. This advantage strongly depends on the reliability of transmission: each lost or modified packet diminishes the effective security level that *Whips* can reach for directly preceding and succeeding received packets. We hence want to investigate the effect of errors on the actual security level provided by our scheme.

The empirical performance assessment (Sec. 6.2) provided only uncontrolled and rare occurrences of errors and are therefore unsuitable for a thorough analysis. Hence, we used synthetic models to analyze the impact of errors on our proposed constructions.

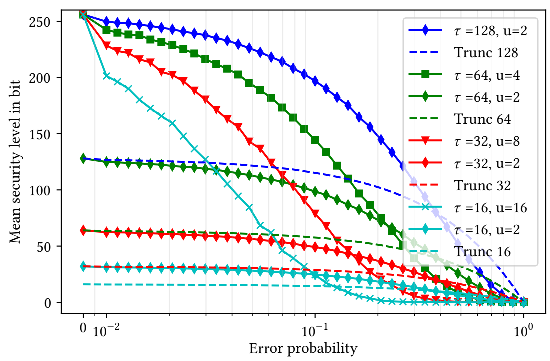


Figure 6: Performance of *Whips* with respect to packet error probability, where an error might be loss of verification failure.

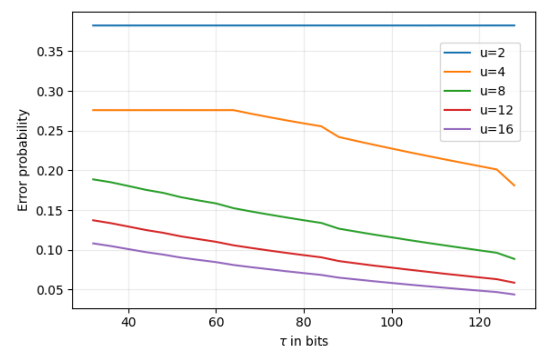


Figure 7: The respective highest error probability, for which the proposed schemes achieve higher security levels than simple truncation. The repetitions yielded maximal standard deviations of 0.00034

A uniformly distributed error represents the worst case scenario, as it reduces the guaranteed security level the most (see Fig. 4). For the subsequent analyses we therefore assume such an error distribution. Thereby, we can deduce the respective bounds for the worst case, whereas practical realizations of the construction are bound to perform better.

For this analysis we simulated 1.000.000 messages transmissions and generated uniformly distributed errors with probability p. Each experiment was repeated 1.000 times to generate statistically sound results. Our schemes are parameterized regarding the tag length τ and the Area of Dependency u, the internal state as well as the key have a size of 256 bits. We compare our schemes against HMAC-SHA256, truncated to the same tag length τ— thereby, our experiments and the compared baseline would achieve to same throughput/speed. As the message length does not influence the achieved security level, we omitted it from this analysis.

The proposed *Whips* construction reacts differently to errors than related schemes, like duplex constructions, as described in Sec. 4: Duplex constructions employ a tightly coupled state chaining. Thus, a single error breaks the transported trust and the chain needs to be explicitly restarted. Although such schemes can always guarantee the full possible security level, the explicit restarting inflicts additional communication overhead, which tends to be unfavourable in the described use cases.

Whips allows for resynchronization after verification errors, shown in Fig. 4. As the effective security level depends on the error occurrences, we analyzed the achieved security level in relation to the error probability after verification of the last message (in Fig. 4 this corresponds to the last time slot, i.e the right-most column).

The results for *Whips* are shown in Fig. 6, plotted in log-scale (we manually inserted the result for a 0% error rate for convenience). Here, an error might be a packet loss or verification failure — thereby, the notion of *mean security level* becomes reasonable: as we are specifically investigating to overall security of the whole message stream, including the mentioned verification failures, the average security level can represent the schemes performance much better. The usually used notation of minimum security would always give 0 *bit* as result in this setting and thus allow little insight.

In this experiment, higher error probability mean more received packets with verification failure (i.e. security level of 0), leading to decreasing mean security levels (even for simple truncation). In the figure different tag lengths τ are represented in identical colors, with the respective baseline being plotted as a dashed line. Values for the same Area of Dependency u are denoted using identical markers. Error probabilities from $0.00 - 0.01$ are plotted linearly, the remaining range up to probability 1 is plotted in log-scale. The figure shows, that for expected and even a range of higher error probabilities our construction can guarantee significantly higher security levels than simple truncation, achieving even up to the full security of 256 bits. For error probabilities of 0.10, the ProMAC construction achieved almost twice the security level as compared to simple truncation (e.g. 128 *bit* vs 211 *bit*). Even when the error probability reaches 0.38, *Whips* still outperforms simple truncation in terms of achieved security.

To understand the actual benefit over truncation better, we investigated the boundary error probability, after which truncation achieves better security than ProMACs. Fig. 7 depicts these results for different combinations of τ and u. Here, the Y-Axis denotes until which upper bound of error probability ProMACs guarantee higher security levels than truncation to τ bits (at identical cost). We observe that with increased Area of Dependency u the boundary drops: the reason of course is that resynchronization requires the verification of u messages, so increasing packet loss has more severe effects on schemes with a large Area of Dependency. The results drop with increasing τ, because there is no advantage in continuing to enlarge the Area of Dependency as the maximum achievable security level is bounded by the length of the state/key.

Packet loss studies of current wireless transmission standards, e.g. 802.15.4 and 802.11, show that packet loss in realistic scenarios can almost be neglected, being at or around 0% in general cases, and rise to under $< 10\%$ even in real-time scenarios [55, 56]. ProMACs can be concluded to reach superior effective security at identical cost in all realistic scenarios.

6.4 Computation, Energy, Storage and Latency

We finally assess computational and storage overhead and theoretical gains.

Computation. Since *Whips* uses two separate PRFs for the calculation of the state update and the tag generation, it inherently adds computational overhead. Consider the aforementioned HMAC as underlying PRF, the usage of *Whips* adds another call this primitive.

By taking the additional state to be hashed into account, the computation changes as follows: The internal state of 168 Bytes (cmp. Sec. 5.4) in the robot control scenario is assumed to be larger than payload messages. Thus, we measured execution of the HMAC-SHA256 for the 15 Byte messages compared to HMAC-SHA512 needed for *Whips* Further, we measured the execution time of the second PRF call on the state, i.e. HMAC-SHA512 of the 168 Bytes substate. All experiments were repeated 1.000.000 times on current hardware (Intel i5 at 2.3GHz).

By employing a modern hash function, SHA3-512, the execution of the first call was equal for SHA3-512 and SHA3-256, with 5.2 μs ($\sigma = 0.33\,\mu s$). The subsequent PRF call generating the final tag took 6.1 μs ($\sigma = 0.31\,\mu s$).

Hence, we conclude that although we are adding a second PRF call, the absolute computational overhead in terms of computation time for the proposed scheme can be considered negligible.

Storage. In terms of required storage, our solution is lightweight and very easy to integrate into the intended use cases. Assuming the aforementioned state of 168 $Bytes$ and a large Area of Dependency of 20, *Whips* requires 3.4 kB state storage. This can be handled very well, even by constrained LoRaWAN sensors which, controlled by STM32L4 chips usually, have 40 to 320 kB RAM available [1].

Energy. Energy overhead is especially relevant for battery-powered devices. Considering the Tmote Sky, the radio unit consumes an order of magnitude more power than the micro controller, at around 4.762 mJ for PHY access and $\approx 1\,\mu J$ for each transmitted Byte [16].

Potential energy savings of *Whips* due to reduced transmission cost range between 21% for 128-bit tags, 31% for 64-bit tags, and as much as 37% for 32-bit tags in the robot use case, as described above. Such gains and the avoidance of retransmissions through resynchronization would directly reduce battery drain, and hence extend the lifetime of motes.

Latency. Considering the robot control use case, even transmission of a normal 20 Byte MAC at 250kbps yields a payload size of only 10 Bytes, ignoring all headers, to achieve the 1ms delay requirement with 802.15.4. Applying *Whips*, this payload size is almost tripled to 27 Bytes, or: considering a message size of 15 Bytes, a 32-bit tag size allows transmission within .6 ms, at the same or higher level of security.

Finally, we note that *Whips* can be used as drop in replacements for existing MAC schemes.

7 SUMMARY

In this paper we tackle the challenge of improving the performance of integrity checking streamed messages, directly upon reception. We introduce the concept of progressive MACs, or ProMACs, to provide security for drone and robot control, distributed control loops in Tactile Internet applications, as well as communication and storage in resource restricted environments. ProMACs integrate the concepts of truncation for performance and state-chaining for increased security while simultaneously exposing inherent resynchronization capabilities. We introduce a unique state update function that facilitates progressive verification of a message upon reception of subsequent tags. The combination allows to significantly reduce overhead while maintaining high levels of security and offering resynchronization.

We present a new construction that realizes ProMACs. Within the messages stream, this scheme constructs a tightly bound trust chain over the internal states while only transmitting a short tag, which allows for efficient transmission like truncated MACs, but guarantees security at the same level as full length MACs. By introducing flexibility regarding the number of incorporated previous states through the Area of Dependency, we allow for the resynchronization after verification errors or packet loss. Here, the Area of Dependency u does not cover all preceding messages, but only those that fall into a sliding window over the stream, as defined by a system parameter. This facilitates resynchronization after u messages have been verified subsequent to an error.

The presented ProMAC construction is suitable to directly replace current MAC schemes. Especially as replacement for shortened tag systems, this delivers the same performance with significantly increased security.

We formally define the construction and prove its respective security and the security levels it can guarantee.

It is worth noting that the formalization of ProMACs extends the current MAC notion by compromising between the different types of statefulness: Our notion covers classical MACs as well as the tight statefulness of duplex based constructions by setting the Area of Dependency respectively.

Finally, we conducted an extensive empirical evaluation demonstrating the applicability of the construction, the performance gains for realistic scenarios, and the effective security levels that are achieved under various packet error probabilities. Thus we show that the ProMACs satisfy all requirements of the described use cases.

In continuation of this work, we currently incorporate the ProMAC design in its presented realization into existing systems as a drop in replacement. Thereby, the impact of ProMACs on the respective use cases can be further evaluated and the corresponding applications can be provided with effective integrity protection.

We also plan further studies on the inherent semantics of the realized security levels, ultimately providing interpretations in various contexts. This will give a more concise meaning to the current confidence into the integrity of received packets and allow for the development of suitable reactions.

ACKNOWLEDGMENTS

This work is partly supported by the German Research Foundation (DFG) in the EXC 2050/1 “CeTI” – ID 390696704. We thank Yannic Ahrens for extensive support with our experimental setup and implementation. We further like to thank the reviewers for their helpful and constructive feedback.

REFERENCES

[1] [n.d.]. *STM32L4 - ARM Cortex-M4 ultra-low-power MCUs - STMicroelectronics.* https://www.st.com/en/microcontrollers-microprocessors/stm32l4-series.html

[2] Ferran Adelantado, Xavier Vilajosana, Pere Tuset-Peiro, Borja Martinez, Joan Melia-Segui, and Thomas Watteyne. 2017. Understanding the limits of LoRaWAN. *IEEE Communications Magazine* 55, 9 (2017).

[3] C. Bachhuber, E. Steinbach, M. Freundl, and M. Reisslein. 2018. On the minimization of glass-to-glass and glass-to-algorithm delay in video communication. *IEEE Transactions on Multimedia* (2018).

[4] M. Bellare, R. Canetti, and H. Krawczyk. 1996. Keying Hash Functions for Message Authentication. In *CRYPTO*. 1–15.

[5] Mihir Bellare, Oded Goldreich, and Anton Mityagin. 2004. The Power of Verification Queries in Message Authentication and Authenticated Encryption. Cryptology ePrint Archive, Report 2004/309. https://eprint.iacr.org/2004/309.

[6] M. Bellare, R. Guérin, and P. Rogaway. 1995. XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions. In *CRYPT0*.

[7] Mihir Bellare, Joe Kilian, and Phillip Rogaway. 2000. The Security of the Cipher Block Chaining Message Authentication Code. *J. Comput. System Sci.* 61, 3 (Dec. 2000), 362–399. https://doi.org/10.1006/jcss.1999.1694

[8] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. 2011. Duplexing the sponge: single-pass authenticated encryption and other applications. In *International Workshop on Selected Areas in Cryptography*. Springer, 320–337.

[9] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. 2013. Keccak. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 313–314.

[10] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway. 1999. UMAC: Fast and Secure Message Authentication. In *CRYPTO*. 216–233.

[11] J. Black and P. Rogaway. 2002. A Block-Cipher Mode of Operation for Parallelizable Message Authentication. In *EUROCRYPT*.

[12] Colin Boyd, Britta Hale, Stig Frode Mjølsnes, and Douglas Stebila. 2015. *From Stateless to Stateful: Generic Authentication and Authenticated Encryption Constructions with Application to TLS.* Technical Report 1150. https://eprint.iacr.org/2015/1150

[13] Ran Canetti and Hugo Krawczyk. 2001. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In *Advances in Cryptology — EUROCRYPT 2001 (Lecture Notes in Computer Science)*, Birgit Pfitzmann (Ed.). Springer Berlin Heidelberg, 453–474.

[14] Y. Challal, H. Bettahar, and A. Bouabdallah. 2004. A2 Cast: An Adaptive Source Authentication Protocol for Multicast Streams. In *Proceedings of the ISCC*.

[15] Y. Challal, A. Bouabdallah, and Y. Hinard. 2005. RLH: Receiver Driven Layered Hash Chaining for Multicast Data Origin Authentification. *Computer Communications* (2005).

[16] Salar Chamanian, Sajjad Baghaee, Hasan Ulusan, Özge Zorlu, Haluk Külah, and Elif Uysal-Biyikoglu. 2014. Powering-up wireless sensor nodes utilizing rechargeable batteries and an electromagnetic vibration energy harvesting system. *Energies* 7, 10 (2014), 6323–6339.

[17] Joan Daemen, Seth Hoffert, G Van Assche, and R Van Keer. 2018. The design of Xoodoo and Xoofff. (2018).

[18] Oliver Eikemeier, Marc Fischlin, Jens-Fabian Götzmann, Anja Lehmann, Dominique Schröder, Peter Schröder, and Daniel Wagner. 2010. History-free aggregate message authentication codes. In *International Conference on Security and Cryptography for Networks*. Springer, 309–328.

[19] ETSI Technical Committee on Electromagnetic compatibility and Radio spectrum Matters. 2011. *Technical characteristics for SRD equipment for wireless industrial applications using technologies different from Ultra-Wide Band (UWB).* Technical Report 102 889-2.

[20] Marc Fischlin, Felix Günther, Giorgia Azzurra Marson, and Kenneth G. Paterson. 2015. Data Is a Stream: Security of Stream-Based Channels. In *Advances in Cryptology – CRYPTO 2015 (LNCS)*. Springer, 545–564.

[21] S. Frankel and H. Herbert. 2003. The AES-XCBC-MAC-96 Algorithm and Its Use With IPsec. RFC 3566.

[22] A. Frotzscher et al. 2014. Requirements and current solutions of wireless communication in industrial automation. In *IEEE International Conference on Communications (ICC)*.

[23] R. Gennaro and P. Rohatgi. 1997. How to Sign Digital Streams. In *Advances in Cryptology — CRYPTO*. 180–197.

[24] Wireless Personal Area Network Working Group. 2016. *802.15.4 - Standard for Low-Rate Wireless Networks.* IEEE.

[25] Shay Gueron. 2016. Memory encryption for general-purpose processors. *IEEE Security & Privacy* 14, 6 (2016).

[26] Mike Hamburg. 2017. The STROBE protocol framework. *IACR Cryptology ePrint Archive* 2017 (2017).

[27] T. Iwata and K. Kurosawa. 2003. Stronger Security Bounds for OMAC, TMAC, and XCBC. In *INDOCRYPT*. 402–415.

[28] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. 2011. *On the Security of TLS-DHE in the Standard Model.* Technical Report 219. https://eprint.iacr.org/2011/219

[29] Jonathan Katz and Andrew Y Lindell. 2008. Aggregate message authentication codes. In *Topics in Cryptology–CT-RSA 2008*. Springer, 155–169.

[30] Jonathan Katz and Yehuda Lindell. 2007. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series).* Chapman & Hall/CRC.

[31] J. Kelsey, S-J Change, and R. Perlner. 2016. *SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash and ParallelHash.* Number NIST SP 800-185. National Institute of Standards and Technology.

[32] Tadayoshi Kohno, Adriana Palacio, and John Black. 2003. *Building Secure Cryptographic Transforms, or How to Encrypt and MAC.* Technical Report 177. https://eprint.iacr.org/2003/177

[33] Vladimir Kolesnikov, Wonsuck Lee, and Junhee Hong. 2011. MAC aggregation resilient to DoS attacks. In *2011 IEEE International Conference on Smart Grid Communications (SmartGridComm)*. IEEE, 226–231.

[34] A. Konstantinos, X. Xu, E. Steinbach, T. Mahmoodi, and M. Dohler. 2018. Towards haptic communications over the 5G Tactile Internet. *IEEE Communications Surveys and Tutorials* (2018).

[35] H. Krawczyk, M. Bellare, and R. Canetti. 1997. HMAC: Keyed-Hashing for Message Authentication. RFC 2104. Updated by RFC 6151.

[36] C. Madson and R. Glenn. 1998. The Use of HMAC-MD5-96 within ESP and AH. RFC 2403.

[37] C. Madson and R. Glenn. 1998. The Use of HMAC-SHA-1-96 within ESP and AH. RFC 2404.

[38] D. McGrew and E. Rescorla. 2010. Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP). RFC 5764.

[39] S. Moriam, E. Franz, P. Walther, A. Kumar, T. Strufe, and G Fettweis. 2018. Protecting Communication in Many-Core Systems against Active Attackers. In *Proceedings of the Great Lakes Symposium on VLSI*. 45–50.

[40] S. Myneni and D. Huang. 2010. IEEE 802.11 Wireless LAN Control Frame Protection. In *IEEE Consumer Communications and Networking Conference*.

[41] National Institute of Standards and Technology. 2005. *NIST special publication 800-38B, Recommendation for block cipher modes of operation: The CMAC mode for authentication.* US Dept. of Commerce.

[42] National Institute of Standards and Technology. 2008. *FIPS 186-4, Digital Signature Standard (DSS).* US Dept. of Commerce.

[43] National Institute of Standards and Technology. 2008. *FIPS 198-1, The Keyed-Hash Message Authentication Code (HMAC).* US Dept. of Commerce.

[44] A. Perrig, R. Canetti, J. D. Tygar, and D. Song. 2000. Efficient Authentication and Signing of Multicast Streams over Lossy Channels. In *Proceedings of IEEE Security and Privacy*. 56–73.

[45] D. E. Phillips, M. M. Moazzami, G. Xing, and J. M. Lees. 2017. A Sensor Network for Real-Time Volcano Tomography: System Design and Deployment. In *International Conference on Computer Communication and Networks (ICCCN)*. 1–9.

[46] Phillip Rogaway and Yusi Zhang. 2018. Simplifying Game-Based Definitions. In *Advances in Cryptology – CRYPTO 2018 (Lecture Notes in Computer Science)*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer International Publishing, 3–32.

[47] Markku-Juhani O Saarinen. 2014. Beyond modes: Building a secure record protocol from a cryptographic sponge permutation. In *Cryptographers' Track at the RSA Conference*. Springer, 270–285.

[48] Guntram Scheible, Dacfey Dzung, Jan Endresen, and Jan Erik Frey. 2007. Unplugged But Connected Design and Implementation of a Truly Wireless Real-Time Sensor/Actuator Interface. *Industrial Electronics Magazine, IEEE* 1 (02 2007), 25 – 34. https://doi.org/10.1109/MIE.2007.901481

[49] J. Schmandt, A. T. Sherman, and N. Banerjee. 2017. Mini-MAC: Raising the bar for vehicular security with a lightweight message authentication protocol. *Vehicular Communications* (2017), 188 – 196.

[50] H. Schweppe, Y. Roudier, B. Weyl, L. Apvrille, and D. Scheuermann. 2011. Car2X Communication: Securing the Last Meter - A Cost-Effective Approach for Ensuring Trust in Car2X Applications Using In-Vehicle Symmetric Cryptography. In *2011 IEEE Vehicular Technology Conference (VTC)*. 1–5.

[51] JH. Song, R. Poovendran, and J. Lee. 2006. The AES-CMAC-96 Algorithm and Its Use with IPsec. RFC 4494.

[52] E. Steinbach, S. Hirche, M. Ernst, F. Brandi, R. Chaudhari, J. Kammerl, and I. Vittorias. 2012. Haptic Communications. *Proc. IEEE* (2012).

[53] JTC 1/SC 27 IT Security techniques. 2011. *Information technology- Security techniques - Message Authentication Codes (MACs)- Part 1: Mechanisms using a block cipher.* ISO/IEC.

[54] JTC 1/SC 27 IT Security techniques. 2011. *Information technology- Security techniques - Message Authentication Codes (MACs)- Part 2: Mechanisms using a dedicated hash function.* ISO/IEC.

[55] Gilles Thonet, Patrick Allard-Jacquin, and Pierre Colle. 2008. Zigbee-wifi coexistence. *Schneider Electric White Paper and Test Report* (2008).

[56] Yi-Hung Wei, Quan Leng, Song Han, Aloysius K Mok, Wenlong Zhang, and Masayoshi Tomizuka. 2013. RT-WiFi: Real-time high-speed communication protocol for wireless cyber-physical control applications. In *2013 IEEE 34th Real-Time Systems Symposium*.