BP-MAC: Fast Authentication for Short Messages

Eric Wagner
eric.wagner@fkie.fraunhofer.de
Fraunhofer FKIE
RWTH Aachen University

Martin Serror
martin.serror@fkie.fraunhofer.de
Fraunhofer FKIE

Klaus Wehrle
wehrle@comsys.rwth-aachen.de
RWTH Aachen University
Fraunhofer FKIE

Martin Henze
henze@cs.rwth-aachen.de
RWTH Aachen University
Fraunhofer FKIE

ABSTRACT

Resource-constrained devices increasingly rely on wireless communication for the reliable and low-latency transmission of short messages. However, especially the implementation of adequate integrity protection of time-critical messages places a significant burden on these devices. We address this issue by proposing BP-MAC, a fast and memory-efficient approach for computing message authentication codes based on the well-established Carter-Wegman construction. Our key idea is to offload resource-intensive computations to idle phases and thus save valuable time in latency-critical phases, *i.e.*, when new data awaits processing. Therefore, BP-MAC leverages a universal hash function designed for the *bitwise preprocessing* of integrity protection to later only require a few XOR operations during the latency-critical phase. Our evaluation on embedded hardware shows that BP-MAC outperforms the state-of-the-art in terms of latency and memory overhead, notably for small messages, as required to adequately protect resource-constrained devices with stringent security and latency requirements.

CCS CONCEPTS

• **Security and privacy → Hash functions and message authentication codes**.

KEYWORDS

message authentication, universal hashing, cyber-physical systems

ACM Reference Format:
Eric Wagner, Martin Serror, Klaus Wehrle, and Martin Henze. 2022. BP-MAC: Fast Authentication for Short Messages. In *Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '22), May 16–19, 2022, San Antonio, TX, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3507657.3528554

1 INTRODUCTION

Cyber-Physical Systems (CPSs), such as industrial control systems, power grids, or smart transportation, depend on mission-critical machine-to-machine communication, where short sensor and control messages need reliable transmission within a few milliseconds and below [11, 18]. The wireless exchange of these messages is extremely challenging, despite being as short as a single byte [11], as it has to account for a shared and error-prone transmission medium [22]. Implementing adequate security for such critical communication further impedes reaching the latency requirements due to the high processing times of cryptographic algorithms on resource-constrained devices [5, 13]. While combining precomputed keystreams with message payload hardly introduces any communication latency for encryption [13], data authenticity and integrity cannot be solved as easily. However, the lack of integrity protection in these scenarios facilitates attacks with severe consequences, ranging from financial losses to threats to human lives [5].

A well-established method for authenticity and integrity protection is a Message Authentication Code (MAC): By appending authentication tags (short: *tags*) to each message, the sender enables the receiver to verify that a message has not been altered during transmission [8]. However, as these tags depend on the transmitted message, their computation and verification introduce delays during the latency-critical phase, *i.e.*, after new data becomes available at the sender and before the receiver can process it. Since CPSs rely on resource-constrained special-purpose devices, *e.g.*, sensors and actuators, they are typically not capable of computing and verifying tags without significant delay [5]. Thus, the delay introduced by MACs must be reduced to enable secure wireless communication even under the most stringent latency requirements.

Therefore, we propose *Bitwise Precomputable MAC* (BP-MAC) to speed up integrity protection by bitwise taking advantage of the characteristics of CPS scenarios: While messages are often only a few bytes long, special-purpose devices are usually idling until new data has to be sensed and transmitted or received and processed [2, 13]. Hence, BP-MAC offloads message-independent and resource-intensive computations to a preprocessing phase where devices are otherwise idle. Moreover, as a Carter-Wegman construction [9, 25], BP-MAC's security can be reduced to traditional time-proven cryptographic primitives and can be trusted to protect even the most sensitive and critical communication against manipulations.

Contributions. We thus present the following contributions:

- We propose BP-MAC, a novel Carter-Wegman scheme requiring only a few XOR operations during the latency-critical phase. It precomputes and caches authentication data for each bit of a message and securely combines them to compute new tags.

WiSec '22, May 16–19, 2022, San Antonio, TX, USA.

ACM ISBN 978-1-4503-9216-7/22/05…$15.00
https://doi.org/10.1145/3507657.3528554

- Since caching comes with a significant memory trade-off, we introduce memory optimizations realizing a smaller memory footprint than BP-MAC's closest contender (UMAC [15]) for messages smaller than 12 to 43 bytes, depending on tag lengths.
- We evaluate BP-MAC's performance on different processor architectures using the Zolertia RE-Mote and Z1 boards to show BP-MAC's capability to reduce the overhead of integrity protection by more than an order of magnitude for small messages.

Availability Statement. The source code underlying this paper is available at: https://github.com/fkie-cad/bpmac

2 MESSAGE AUTHENTICATION CODES

In the following, we formally introduce Message Authentication Codes (MACs) (Sec. 2.1) and universal hashing (Sec. 2.2) as a basis for efficient, secure authentication schemes. Then, we discuss related work on optimizing MACs for low-latency scenarios (Sec. 2.3).

2.1 Formal Definition

A MAC scheme is composed of two algorithms, Sig_k and $Vrfy_k$, that generate and verify authentication tags t with the help of a pre-shared secret key k [8]. A message m, extended by a tag t, computed as $t = Sig_k(m)$, allows the receiver of a message to verify the integrity of the received message. Upon reception of a message-tag pair (m, t), the receiver uses the algorithm $Vrfy_k(m, t)$ to verify whether the message or the tag has been manipulated in transit by an attacker not possessing the secret key k. In most popular MAC schemes, $Vrfy_k(m, t)$ computes the tag $t^* = Sig_k(m)$ based on the received message m and returns whether $t^*$is identical to the received tag t. Consequently, MAC schemes are considered secure if, even under a chosen-message attack, an attacker cannot create an existential MAC forgery, *i.e.*, a tag t for a previously unseen message m, that $Vrfy_k(m, t)$ would accept. Achieving this property typically requires the execution of (for resource-constrained devices) processing-intensive cryptographic algorithms.

2.2 The Carter-Wegman MAC Construction

The Carter-Wegman construction allows building efficient and secure MAC schemes, such as UMAC [15] and Poly1305 [6], which gained popularity due to their superior performance compared to traditional schemes. Carter-Wegman MAC constructions add an additional nonce n to the parameters of Sig_k and $Vrfy_k$, which natively protects against replay attacks [8]. These constructions first compute the digest of the message m with a hash function unknown to the attacker. Then, this digest is masked by the encrypted nonce. More formally, a Carter-Wegman MAC construction computes the tag t_i for a message m_i and a nonce n_i as $t_i = F_{k1}(m_i) \oplus H_{k2}(n_i)$, where H is a pseudorandom function covering the same output space as the universal hash function F [8]. A universal hash function is a (not necessarily cryptographic) secret hash function for which it is hard to find a collision, *i.e.*, finding m, m' such that $F(m) = F(m')$, as long as the attacker does not know a single output of F [8].

2.3 Efficient MACs for Low Latency Scenarios

Optimized computations of established primitives such as AES or SHA256 have been proposed to address the need for fast message authentication on resource-constrained devices. Such approaches include hardware acceleration [26] or the preprocessing of authentication for predictable parts of future messages [13]. However, these approaches still fail to enable secure sub-millisecond communication as many CPS scenarios demand [11, 18]. Consequently, a range of novel MAC algorithms based on new cryptographic primitives has been proposed to reduce processing and latency overheads for resource-constrained devices, including SipHash [4], TuLP [12], Chaskey [21], and LightMAC [19]. MergeMAC [2] saves valuable time during latency-critical phases by extracting predictable parts of future messages to authenticate them in advance. However, mission-critical CPS must rely on scrutinized and time-proven cryptographic primitives to ensure the maximum possible security guarantees.

Therefore, competing approaches target efficient Carter-Wegman MAC schemes, whose security can be reduced to underlying cryptographic primitives (*e.g.*, AES). To do so, UMAC [15] and VMAC [14] use custom hash functions optimized for 32-bit and 64-bit architectures, respectively. The universal hash function employed by Poly1305 evaluates a polynomial with the message as coefficients over the finite field $\mathbb{Z}_{2^{130}-5}$ [6]. Although highly efficient, these schemes optimize for messages of at least a few hundred bytes [4]. PMAC [7] fragments messages to parallelize and thus speed up tag computations. Nevertheless, typical resource-constrained devices with single-core processors do not benefit from such optimizations.

Another branch of research tackles the reduction of bandwidth overhead introduced by MACs schemes via progressive MACs (ProMACs [3]). ProMACs reduce the tag sizes by aggregating authentication over several messages [3, 16, 17, 23, 24]. Thereby, ProMACs protect each message with immediately reduced security, allowing their optimistic processing without waiting for the reception of subsequent messages. Still, a message's authenticity is reinforced with subsequent messages to ensure eventual strong security [3]. Like traditional authentication, ProMACs also benefit from faster underlying MAC schemes to reduce processing latency.

3 BP-MAC: A BITWISE PRECOMPUTED MAC

To remove the bottleneck of cryptographic processing for message authentication and thus enable low-latency transmissions of small messages on resource-constrained devices using conventional security primitives, we propose *Bitwise Precomputable MAC* (BP-MAC). BP-MAC combines the idea of preprocessing message-independent computations with a Carter-Wegman MAC construction specifically designed for small messages. We begin by discussing preprocessing as the underlying principle of BP-MAC (Sec. 3.1) before detailing BP-MAC's construction (Sec. 3.2). Then, we present memory optimizations (Sec. 3.3) and discuss the security of BP-MAC (Sec. 3.4).

3.1 Precomputations for Fast Authentication

BP-MAC takes advantage of precomputations since the processing loads in CPS scenarios are not evenly distributed over time [2, 13]. Instead, devices periodically have idle times until new data has to be transmitted, received, or processed. Hence, the idea behind BP-MAC is to offload computationally expensive operations into those idling phases to reduce computations during latency-critical phases to a minimum. Therefore, a naïve approach would be to precompute and store authentication tags for each possible message. However, precomputing authentication information, *e.g.*, for all possible, rather

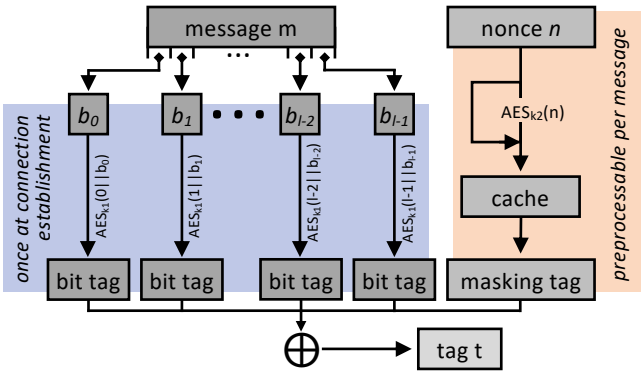

Figure 1: BP-MAC achieves high-speed tag computations during the latency-critical phase by only XORing the bit tag depending on each bit value and the masking tag to generate a secure authentication tag. The masking tag is independent of the message and can thus be precomputed during idle times. Bit tags are precomputed once for each unique key.

tiny two-byte messages would then already consume 1 MB of memory (assuming the recommended tag length of 16 bytes), a number that is exponentially growing for longer messages. Hence, this naïve approach is infeasible for resource-constrained CPS devices.

In contrast, the core idea of BP-MAC is to precompute tags for individual bits, which are then efficiently combined when a message needs authentication. As each bit can only represent either 0 or 1, BP-MAC's memory consumption grows linearly with message lengths and amounts to only 512 bytes for *all* possible two-byte long messages. These tags can be precomputed once a new key is exchanged or even provided by the (more powerful) communication partner. In addition, BP-MAC masks the combined tags with a secret masking tag to prevent replay attacks and enable their secure aggregation. Such tags can be conveniently precomputed between transmissions since they are independent of the actual message.

3.2 BP-MAC's Design in Detail

BP-MAC achieves fast authentication by only executing the efficient aggregation of previously precomputed tags during the latency-critical phase. We illustrate the different steps of BP-MAC's design in Figure 1 and discuss them in detail in the following.

3.2.1 Bit Tags. As shown in Figure 1, bit tags are intermediate tags computed for each message bit. To ensure BP-MAC's security and prevent collisions, all bit tags have to be a unique pseudorandom number that must be kept secret. BP-MAC computes bit tags as the digest of a pseudorandom function by concatenating bit index and bit value. Concretely, BP-MAC uses AES-128$_{k1}$ as a pseudorandom function. Here, $k1$ is a secret key shared between both communicating parties due to its increased efficiency over other functions, *e.g.*, HMAC-SHA256. Thus, if the value of the third bit in the second byte is zero, the corresponding bit tag is AES-128$_{k1}(10||0)$. Overall, there exist $2 \cdot l$-bit tags that have to be precomputed for a message with l bits. However, this only occurs once a new key is established and could even be offloaded to a more powerful device if necessary.

3.2.2 Masking Tags. The purpose of masking tags is twofold. First, they provide replay protection by incorporating a nonce into the final tag. Second, these pseudorandom tags prevent the leakage of information concerning individual bit tags and are thus crucial for the security of BP-MAC. As nonce, BP-MAC uses a zero-initialized counter that is incremented for each newly computed tag.

Computing the masking tag from a pseudorandom function and the nonce is analogous to UMAC [15] and relies on AES-128$_{k2}$. For tags with a length between 9 and 16 bytes, we use the leading n bytes of the AES-encrypted nonce, where $k2$ ($\neq k1$) is also a secret key shared between both parties. For smaller tags, we use the cached 16-byte long AES output for multiple masking tags, as long as no output bytes are reused. Thus, for tags of 4 bytes, only every fourth nonce has to be encrypted to reduce processing overhead, while we can use the unused bytes of the cached output to blind the remaining tags. As nonces can be predicted, masking tags can be precomputed and thus do not add to the latency-critical delay.

3.2.3 Tag Computation and Verification. After discussing the two precomputation phases generating the bit and masking tags, we now explain the computation of the n-th authentication tag t_n for the message m_n. As shown in Figure 1, the final step of tag computation consists of XORing the intermediate tags. This observation also manifests in the formal definition of computing tag t_n:

$$\mathrm{t_n} = \bigoplus_{i \in |m_n|} \underbrace{\text{AES-128}_{\mathrm{k1}}(i||m_n[i])}_{\text{bit tags}} \oplus \underbrace{\text{AES-128}_{\mathrm{k2}}(n)}_{\text{masking tag}}$$

The second part is the masking tag that is XORed with the bit tags that correspond to the message. Here $|m|$ denotes the length of the message and $m[i]$ denotes the value of the i-th bit. Thus, all computations besides these final XOR operations are independent of the message that should be authenticated. Therefore, BP-MAC's construction enables the quick computation of new tags, especially for short messages which require a low number of XOR operations.

As usual for deterministic MAC algorithms, the tag verification happens by computing the tag t^* for the received message m and then comparing it to the received tag t. If both tags are identical, the receiver can conclude, with high confidence, that the received message m has not been altered in transit.

3.3 Memory Optimizations for BP-MAC

The naïve realization of BP-MAC requires the storage of $2 \cdot |m|$ bit tags. For a fixed-length messages, we can, however, half this memory overhead by precomputing the default tag of a message composed of only zeros. To support the same optimization for variable-length messages, we must pad the message before authenticating it. Then, we only need to store how to alter the tag for a bit set to one with so-called *bitflip* tags, of which we need $|m|$ in total (one for each bit). We sketch the tag computation with this optimization in Algorithm 1, highlighting in blue the computations executed during the latency-critical phase. Already before a new message is ready, we can compute the masking tag and XOR it with the default tag $\mathrm{t_{default}}$ to generate the tag of an all-zero message as:

$$\mathrm{t_{default}} = \bigoplus_{0 \le i < n} \text{AES-128}_{\mathrm{k1}}(i||0)$$

Then, to compute the tag for the actual message, we need to change the tag for each bit that is not zero as assumed for the default tag

Algorithm 1: Memory-efficient signature generation Sig^*_k

Input: message *msg*, nonce *nonce*
Output: tag t

```
t ← AES_k2(nonce)                    ▷ Preprocessable masking tag
len ← length(msg)                    ▷ The length of the message in bits
t ← t ⊕ t_default                     ▷ Compute tag of all-zero message
for n between 0 and len-1 do
    if the n-th bit in msg is set then
        t ← t ⊕ t_bitflips[n]        ▷ Lookup the cached change to t
    end
end
t ← t ⊕ t_bitflips[len]               ▷ Padding for the message
return t
```

$t_{default}$. To efficiently realize these changes, we can precompute bitflips tags for all i bits in a message as:

$$t_{bitflip}[i] = \text{AES-128}_{k1}(i||0) \oplus \text{AES-128}_{k1}(i||1)$$

By XORing these bitflip tags to the default tag $t_{default}$, we compute the tag as if the bit at the corresponding position is set. Finally, we pad the message to a fixed length to be able to differentiate between a shorter message and one with trailing zeros. BP-MAC uses the *Padding method 2* as described by ISO/IEC 9797-1 [1], appending a 1-bit at the end of the message followed by as many 0-bits as necessary to reach the desired message length. As the default tag $t_{default}$ already assumes that all bits are zero, we only need to incorporate a single further bitflip tag t for the first index after the end of the message. Thus, we only need to store $|m| + 3$ tags (one for each bit, including padding, t_{nonce}, and $t_{default}$). This procedure reduces the number of operations during the latency-critical phase for each 0-bit, which accounts for about half of encrypted traffic.

3.4 Security Discussion

BP-MAC utilizes the secure Carter-Wegman MAC construction [9, 25] in which a tag t for a message m and a nonce n is computed as $t = F_{k1}(m) \oplus H_{k2}(n)$, where H is a pseudorandom function covering the same output space as the universal hash function F (*cf.* Sec. 2.2). For H, BP-MAC uses the same AES-based procedure as UMAC [15], with the sole difference that it computes the result in advance. Meanwhile, the fragmentation of messages into individually authenticatable bits is a special case of the established universal hash function $F^{\oplus}$ [8]. Thus, BP-MAC is secure as long as the underlying cryptographic primitive (*i.e.*, AES) is secure. Moreover, BP-MAC can easily exchange this primitive, if ever considered not secure enough. In the following, we define the considered threat model and subsequently discuss possible attack scenarios.

3.4.1 Threat Model. The attacker's goal is to alter traffic such that the recipient of a message accepts the modified message is genuine. To achieve this goal, the attacker can observe and alter the (plaintext) messages and exchanged tags to either learn the secret key used by BP-MAC or directly alter a message and the corresponding tag. The information to realize such an attack can be extracted directly from the observed traffic or through side-channel information by observing the timing of individual packets. However, we explicitly do not consider an attacker with (partial) control over one or both communicating entities that could try to access keying information through a *e.g.*, cache side-channel attacks. However, BP-MAC does not prevent the use of common mitigation techniques that protect against such attacks [20].

3.4.2 Resilience to Key Recovery Attacks. BP-MAC is not susceptible to key recovery attacks, as otherwise a key recovery attack against the underlying cryptographic primitive, *i.e.*, AES, would exist. By overhearing transmissions, an attacker learns $t = F^{\oplus}_{k1}(m) \oplus H_{k2}(n)$ for a known message m and nonce n. An attacker cannot learn $k2$ from such tags, as otherwise there would exist a key recovery attack against H_{k2}, *i.e.*, `AES-128`: After learning a digest $d = H_{k2}(m)$, the attacker could generate a key k', consider all unique messages m as nonces, and compute a BP-MAC tag t' for arbitrary new messages m' as $t' = F^{\oplus}_{k'}(m') \oplus H_{k2}(m)$. Thus, if an attack that recovers $k2$ from t exists, this attack could be used to recover $k2$ from simple H_{k2} digests. Similarly, $k1$ cannot be recovered by an attacker. Even if an attacker would learn the output of $F^{\oplus}_{k1}(m)$, a recovery of $k1$ would imply the existence of a key recovery attack against `AES-128`: The attacker can compute $F^{\oplus}_{k1}(m_1||\cdots||m_k)$ for observed ciphertexts $ct = AES_k(pt)$ by considering the first part of an observed plaintext pt to be the index and the second part to be a truncated message, *i.e.*, $pt = i||m_i$. Then, if a key recovery attack would exist against $F^{\oplus}$, this same attack would also enable recovering k for `AES` plaintext-ciphertext pairs.

3.4.3 Unforgeability of Authentication Tags. However, a successful attack does not need to recover the authentication keys to attack a MAC scheme. In many cases, it suffices if an attacker can forge authentication tags and thus inject harmful messages into a protected communication. Nevertheless, BP-MAC is also secure against such attacks. First, a message is hashed by $F^{\oplus}$, with an unpredictable result for the attacker if they did not observe previous digests of $F^{\oplus}$. Therefore, each output of $F^{\oplus}$ is masked by a unique pseudorandom number, ensuring that an attacker cannot learn a single output of $F^{\oplus}$. As the nonce changes for each message, no information about the internal state of BP-MAC is revealed to an outsider. Thus, BP-MAC is also secure in the presence of an attacker that merely wants to generate valid tags for altered or new messages.

3.4.4 Timing Side-Channel Attack against BP-MAC. The presented memory optimizations for BP-MAC execute some computations only for bits set to one, potentially enabling timing side-channel attacks. In particular, the time a message authentication computation takes might hint at the number of bits set to one in the authenticated message. If the message is sent in plaintext, this information is already accessible to third parties and does not leak confidential information or help in recovering keys. However, when authenticating encrypted traffic, it is crucial operating in the *encrypt-then-MAC* mode [8] not to reveal information about the plaintext.

4 PERFORMANCE EVALUATION

BP-MAC reduces the delay of message authentication for short, mission-critical wireless communication. We validate this claim

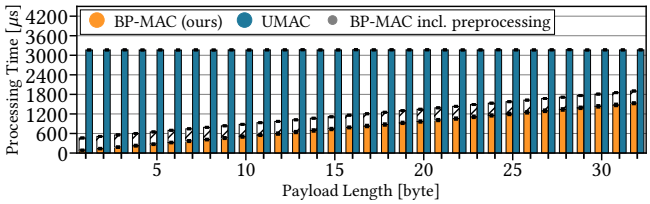

Figure 2: On the 16-bit architecture of the Zolertia Z1, BP-MAC outperforms UMAC by up to two orders of magnitude for short messages and shows significantly less overall processing overhead for 32 byte long messages.

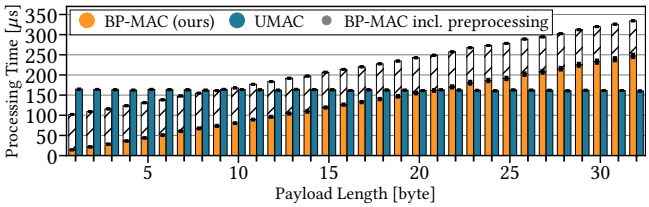

Figure 3: Even on the faster Zolertia RE-Mote, BP-MAC realizes faster tag computations than UMAC in the latency-critical phase for messages shorter than 21 bytes and less overall processing for messages shorter than 10 bytes.

by performing measurements on two different embedded devices (Sec. 4.1) and evaluating BP-MAC's memory consumption (Sec. 4.2).

4.1 Latency Measurements

We implement BP-MAC for Contiki-NG and evaluate its performance on two different architectures to ensure the observed benefits generalize across them: Zolertia Z1 (MSP430 @ 16 MHz, 16-bit CPU, 8 kB RAM) and Zolertia RE-Mote (ARM Cortex-M3 @ 32 MHz, 32-bit CPU, 16 kB RAM). To assess the performance of BP-MAC, we have to compare it to other MAC schemes with similar security guarantees, such as UMAC, VMAC, or Poly1305. We choose to compare BP-MAC against an optimized implementation of UMAC[1] as it introduces the least processing overhead for short messages (<32 bytes) even on an unfavorable architecture [14].

4.1.1 BP-MAC on the Zolertia Z1. First, we compare BP-MAC's and UMAC's performance on the 16-bit MSP430 processor of the Zolertia Z1, a typical architecture for resource-constrained CPS devices. Here, we report on the time for computing 16-byte tags for messages of varying lengths. Figure 2 depicts the means and 99% confidence intervals for the respective computations of one tag. We measured 100 tag computations and derive from this the time of one tag computation due to a too low clock resolution. We repeated each measurement 30 times. Note that Sig_k and Vrfy_k require one tag computation each, such that the actual delay introduced into one secure transmission is twice the reported time.

As expected, the time required to compute UMAC tags is independent of the message length on the analyzed scale. In contrast, BP-MAC's bitwise processing introduces a linear dependency between message length and processing time. We observe that BP-MAC significantly outperforms UMAC, such that even for 32 bytes long messages, the overall processing time of BP-MAC is still 40 % lower. For 1 byte long messages, the difference is even more extreme, as BP-MAC induces a delay of only 86 μs for one tag computation, compared to 3.2 ms for UMAC. BP-MAC is thus significantly faster than UMAC for short messages on a 16-bit architecture.

We repeated the same measurements for shorter tags (12, 8, and 4 bytes). There, we observed similar behavior, whereas the absolute processing time of BP-MAC became shorter by 24.8 – 35.4%, 38.6 – 54.0%, and 51.1 – 66.5% for increasingly shorter tags as shorter tags require less XOR operations. In turn, UMAC also becomes faster, primarily through its ability to reuse AES computations to mask multiple tags for 8 and 4-byte tags. However, the irregular need to compute AES blocks introduces jitter to the processing of UMAC, which is undesirable in low-latency communication [10]. For BP-MAC, these bursty computations only occur in the preprocessing phase and thus do not influence actual transmission delays.

4.1.2 BP-MAC on the Zolertia RE-Mote. To show the performance of BP-MAC on a more powerful device, we furthermore compare BP-MAC and UMAC on the Zolertia RE-Mote. While it still constitutes an embedded device, its ARM Cortex-M3 is significantly more powerful than the Zolertia Z1. Also, its 32-bit processor is precisely the architecture targeted by UMAC. Hence, we repeat our previous measurements and report on the results in Figure 3.

Still, BP-MAC outperforms UMAC for small messages, partially by more than one order of magnitude. To be precise, the latency-critical processing of BP-MAC is faster for messages not longer than 21 bytes. Furthermore, even the overall processing overhead is smaller for messages shorter than 10 bytes. These results thus show that BP-MAC is particularly suited for authenticating short messages, even in a worst-case comparison to UMAC. For shorter tags, this trend continues, with BP-MAC outperforming UMAC by 17.2 – 24.4%, 22.8 – 32.0%, and 35.2 – 48.3% during the latency-critical phase for tags of sizes 12, 8, and 4, respectively. The tipoff points below which BP-MAC outperforms UMAC are 26, 18, and 15-byte messages for increasingly shorter tags.

Concluding, BP-MAC enables secure communication based on the established security of AES with low processing overhead for small messages across different processor architectures. However, the performance gap between BP-MAC and UMAC is much narrower on the Zolertia RE-Mote. This behavior is not due to a worse performance by BP-MAC, but instead due to UMAC being specifically optimized for the 32-bit processor in this scenario.

4.1.3 BP-MAC and Hardware-accelerated Cryptography. Due to the rising importance of low-latency security in CPS scenarios, low-powered devices increasingly provide hardware accelerators for cryptographic operations. However, using the Zolertia RE-Mote's sha-256 accelerator for HMAC computations shows that even hardware-accelerated cryptography can introduce significant processing overheads. Indeed, our measurements reveal that the computation of one HMAC-SHA256 tag takes approximately 220 μs, independent of tag and payload lengths for small payloads. Thus, in these cases, hardware-accelerated HMAC-SHA256 is even slower than a software implementation of UMAC. Still, accelerators for specialized MAC schemes can be constructed to achieve even faster cryptographic operations. Fittingly, BP-MAC's heavy reliance on XOR operations and its high parallelizability by individually processing each bit promise highly efficient hardware implementations.

[1] https://fastcrypto.com/umac

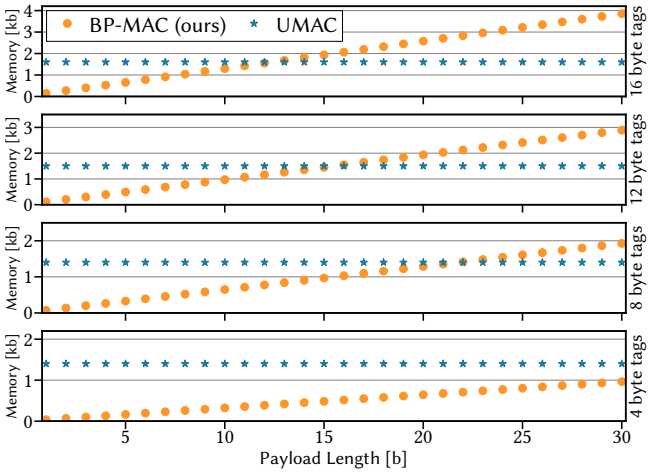

Figure 4: BP-MAC's memory footprint grows linearly with tag and message lengths. Thus, for messages shorter than 12, 15, 21, and 43 bytes, BP-MAC requires less memory than UMAC for 16, 12, 8, and 4-byte long tags, respectively.

4.2 Memory Overhead

Fast MAC schemes based on universal hashing, such as BP-MAC and UMAC, typically trade memory usage for high processing speeds, a limited resource on low-power devices. Hence, we compare BP-MAC's and UMAC's memory footprint by compiling Contiki-NG as a native Linux application. We then use Valgrind's massif tool to analyze the peak memory footprint of both schemes for varying message and tag lengths. We depict our results in Figure 4.

We notice that the memory footprint of UMAC is constant for different message sizes and hardly changes across tag lengths (ranging from 1.4 kB to 1.6 kB). In contrast, BP-MAC's memory footprint increases with the sizes of tags and messages. Overall, BP-MAC's memory footprint increases by eight times the tag lengths when messages become one byte longer, which we expected since an additional bitflip tag has to be stored for each additional message bit. Consequently, BP-MAC's memory footprint is favorable for small messages up until a tipoff point where UMAC becomes, in turn, more resource-efficient. These tipoff points lie at a message size of 12, 15, 21, and 43 bytes for 16, 12, 8, 4 byte long tags, respectively.

Overall, our evaluation thus shows that BP-MAC outperforms UMAC in terms of processing latency and memory footprint for small messages while providing the same security guarantees, *i.e.*, reducible to the security of AES. Thus, BP-MAC enables secure communication in critical CPS scenarios with a significantly smaller impact on communication latency than state-of-the-art approaches.

5 CONCLUSION

CPSs rely on low-latency wireless communications with stringent security guarantees. In this context, a significant challenge is to ensure the authenticity and integrity of exchanged messages, which typically leads to computationally intensive calculations in latency-critical phases, e.g., once a new message is ready for transmission. While numerous MAC schemes aim to reduce latency, none address small messages of only a few bytes, an essential characteristic of many latency-critical CPS scenarios. To fill this gap, we propose a new Carter-Wegman MAC scheme with AES-based security (BP-MAC), optimizing the extensive use of preprocessing with a universal hash function for the aforementioned small messages. Thus, BP-MAC reduces latency-critical computations to a few XOR operations, which overall leads to more than a ten-fold reduction in processing overhead compared to the state-of-the-art. Consequently, BP-MAC enables integrity protection of small messages in latency-critical scenarios, even for CPS devices with limited computational and memory resources.

ACKNOWLEDGMENTS

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC-2023 Internet of Production – 390621612. We thank the reviewers and our shepherd Kasper Rasmussen for their fruitful comments.

REFERENCES

[1] 2011. ISO/IEC 9797-1:2011 Information technology - Security techniques - Message Authentication Codes (MACs) - Part 1: Mechanisms using a block cipher.
[2] Ralph Ankele et al. 2018. MergeMAC: a MAC for authentication with strict time constraints and limited bandwidth. In *ACNS*.
[3] Frederik Armknecht et al. 2020. ProMACs: Progressive and Resynchronizing MACs for Continuous Efficient Authentication of Message Streams. In *ACM CCS*.
[4] Jean-Philippe Aumasson and Daniel J Bernstein. 2012. SipHash: a fast short-input PRF. In *Indocrypt*.
[5] Amira Barki et al. 2016. M2M Security: Challenges and Solutions. *IEEE Communications Surveys & Tutorials* 18, 2 (2016).
[6] Daniel J Bernstein. 2005. The Poly1305-AES message-authentication code. In *FSE*.
[7] John Black and Phillip Rogaway. 2002. A block-cipher mode of operation for parallelizable message authentication. In *Eurocrypt*.
[8] Dan Boneh and Victor Shoup. 2020. A graduate course in applied cryptography.
[9] J Lawrence Carter and Mark N Wegman. 1979. Universal Classes of Hash Functions. *Journal of computer and system sciences* 18, 2 (1979).
[10] Andreas Frotzscher et al. 2014. Requirements and current solutions of wireless communication in industrial automation. In *ICC*.
[11] Brendan Galloway and Gerhard P Hancke. 2012. Introduction to Industrial Control Networks. *IEEE Communications surveys & tutorials* 15, 2 (2012).
[12] Zheng Gong et al. 2014. Tulp: A family of lightweight message authentication codes for body sensor networks. *JCST* 29, 1 (2014).
[13] Jens Hiller et al. 2018. Secure low latency communication for constrained industrial iot scenarios. In *LCN*.
[14] Ted Krovetz. 2006. Message authentication on 64-bit architectures. In *SAC*.
[15] Ted Krovetz et al. 2006. UMAC: Message authentication code using universal hashing. RFC 4418.
[16] He Li et al. 2020. Cumulative Message Authentication Codes for Resource-Constrained Networks. In *CNS*.
[17] He Li et al. 2021. Cumulative Message Authentication Codes for Resource-Constrained IoT Networks. *IEEE Internet of Things Journal* 8, 15 (2021).
[18] Michele Luvisotto et al. 2017. Ultra High Performance Wireless Control for Critical Applications: Challenges and Directions. *IEEE TII* 13, 3 (2017).
[19] Atul Luykx et al. 2016. A MAC mode for lightweight block ciphers. In *FSE*.
[20] Yangdi Lyu and Prabhat Mishra. 2018. A survey of side-channel attacks on caches and countermeasures. *Journal of Hardware and Systems Security* 2, 1 (2018).
[21] Nicky Mouha et al. 2014. Chaskey: an efficient MAC algorithm for 32-bit microcontrollers. In *SAC*.
[22] Mohsin Raza et al. 2018. A Critical Analysis of Research Potential, Challenges, and Future Directives in Industrial Wireless Sensor Networks. *IEEE Communications Surveys & Tutorials* 20, 1 (2018).
[23] Jackson Schmandt et al. 2017. Mini-MAC: Raising the bar for vehicular security with a lightweight message authentication protocol. *Vehicular Communications* 9 (2017).
[24] Eric Wagner et al. 2022. Take a Bite of the Reality Sandwich: Revisiting the Security of Progressive Message Authentication Codes. In *WiSec*.
[25] Mark N Wegman and J Lawrence Carter. 1981. New Hash Functions and Their Use in Authentication and Set Equality. *J. Comput. Syst. Sci* 22, 3 (1981).
[26] Kaiyuan Yang et al. 2017. Hardware designs for security in ultra-low-power IoT systems: An overview and survey. *IEEE Micro* 37, 6 (2017).