MergeMAC: A MAC for Authentication with Strict Time Constraints and Limited Bandwidth

Ralph Ankele[1]*, Florian Böhl[2], Simon Friedberger[2]*

[1] Royal Holloway University of London, United Kingdom.
ralph.ankele.2015@rhul.ac.uk
[2] NXP Semiconductors, Belgium.
{florian.boehl, simon.friedberger}@nxp.com

Abstract. This paper presents MergeMAC, a MAC that is particularly suitable for environments with strict time requirements and extremely limited bandwidth. MergeMAC computes the MAC by splitting the message into two parts. We use a pseudorandom function (PRF) to map messages to random bit strings and then merge them with a very efficient keyless function. The advantage of this approach is that the outputs of the PRF can be cached for frequently needed message parts. We demonstrate the merits of MergeMAC for authenticating messages on the CAN bus where bandwidth is extremely limited and caching can be used to recover parts of the message counter instead of transmitting it. We recommend an instantiation of the merging function Merge and analyze the security of our construction. Requirements for a merging function are formally defined and the resulting EUF-CMA security of MergeMAC is proven.

Keywords: Symmetric-key cryptography, message authentication code, lightweight, efficient, automotive, CAN bus

1 Introduction

In constrained environments, such as embedded devices, wireless sensor networks, control systems and automated devices, the Internet of Things (IoT) in general, and more particularly to save bandwidth in highly time constrained scenarios it is important to reduce the latency while still ensure that the throughput of a communication channel does not fall below a critical threshold. A common practice is to omit parts with low entropy in the communication of transmitted messages when the communication overhead is more expensive than simply reconstructing the message by brute-force guessing the missing parts. As attacks on IoT devices become more and more of a threat [Lan11] [RSWO17], it is important to ensure the authenticity of an entity and the integrity of transmitted messages. Message Authentication Codes (MAC) allow an receiver of a message

* This research was partially supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No. H2020-MSCA-ITN-2014-643161 ECRYPT-NET.

to verify the integrity of a received message and ensure that the sender has to be an authenticated entity.

MergeMAC is a MAC function optimized exactly for the requirements of communicating in extremely constrained environments and still ensuring strict time requirements even with limited bandwidth. In our proposal, we omit parts of the transmitted message with very little information content. The missing information can then be reconstructed by the receiver by brute-forcing all possibilities as only the correct solution will be verified as the received MAC. Even though the missing parts are not transmitted they are included in the MAC computation. Our proposed construction achieves the strict timing constraints, while still recalculating parts of the MAC, by splitting the MAC computation into several parts, and further allowing fast recombination of intermediate values into MACs. We will now introduce the scenario which prompted the development of our solution.

Security under extreme conditions. Modern cars rely on the Controller Area Network (CAN) bus for car-internal communication between different components, usually referred to in this context as electronic control units (ECUs). The CAN bus connects everything - from critical systems like breaks or airbags to convenience features like built-in navigation or entertainment systems. When the bus was developed in the 1980's, the connected world of today was still merely science fiction. Security was not a concern. This started to change when hackers first used the CAN bus as an entry point to steal a car, accessing it by taking off a side mirror [KCR+10,CMK+11]. Nowadays, more and more components with an interface to the CAN bus are also connected to the Internet. This allows hackers to mount a powerful attack, by attacking the car from an online attack surface. A remote attack on the complete fleet is the nightmare of every CTO in the automotive industry. Consequentially, the call for more security grew louder quickly. However, adding security to legacy systems is a complicated and error prone process. The ramifications specifically for the CAN bus are summarized as:

- **High costs:** The cost pressure is high. Many ECUs are simple microcontrollers and a security solution that adds significant costs is unlikely to be adopted.
- **Time constraints:** Several components, such as breaks or airbags, are running under strict time constraints. This confines the latency a security solution may introduce to the system.
- **Limited bandwidth:** The bandwidth on the CAN bus is limited. Hence a security solution should need as little additional bandwidth as possible.

In this paper we describe an approach how to establish authenticated communication between ECUs on the CAN bus under these extreme conditions. Our construction allows for very efficient hardware implementations as we demonstrate with the specific instantiation based on MergeMAC and Present or Prince. It addresses bandwidth constraints by not transmitting low-entropy parts of messages and recovering them at the receiver in an efficient way.

Message Authentication Codes. Message Authentication Codes (MACs) are a popular standard technique to authenticate messages using a shared secret

key [ISO11][Dwo16]. Using a shared key and a message as input, a user can compute a MAC tag and attach it to the message. The recipient can then verify the authenticity of the message using the secret key. The most striking advantage of MACs is their efficiency. MACs can be constructed directly from symmetric block ciphers or from cryptographic hash functions and are hence much more efficient than digital signatures, their asymmetric relatives. This also makes them the tool of choice on constrained devices, like microcontrollers and lightweight IoT devices. Therefore, they are the correct building blocks for our scenario.

Authentication. The CAN bus itself does not provide any message authentication mechanism. Every connected ECU can send arbitrary frames including a forged sender identity. Our goal is to define an authentication mechanism that is tailored towards the most common scenario: A sender (e.g. a sensor) addressing one or more receivers. The sender includes a counter into each message to prevent replay attacks and adds a MAC tag to each message sent. Receivers keep track of the counter value. They will reject incoming messages with a counter value not higher than the current one. Skipping messages is acceptable, i.e. the counter value will always be updated to the last one received with a valid MAC tag. The authentication mechanism we propose significantly reduces the attack surface if it is used for all ECUs on the CAN bus. Moreover, the attack surface is drastically reduced, as an attacker from the outside (e.g. by accessing the CAN bus from the side mirrors) will not be able to impersonate an internal device. Furthermore, an attacker from the inside (e.g. through a corrupted ECU) is confined to the receivers that trust this ECU.

A bandwidth saving technique. Bandwidth can be saved by only including the n least significant bits of the counter in the messages, i.e. the counter is divided into n low bits, called l and the remaining high bits called h. The sender computes the MAC tag on the message as usual based on the complete counter value. The receiver recovers the complete counter as follows: He takes the high bits h from his local storage and combines them with the transmitted low bits l. If the MAC verification is successful and the combined counter is greater than the locally stored counter value, the message is accepted. Otherwise, the message may still be valid if a wrap of the lower bits l has occurred on the sender side leading to an incremented h. Consequentially, the receiver will try again with $h+1$, $h+2$, ... up to $h+w_{max}$ for a parameter w_{max} of the maximum number of wraps that are allowed to occur. We apply this idea to the counter only, but it can be used more broadly to recover predictable information to save bandwidth. With 8 bytes, payload sizes on the standard CAN bus are relatively short and we will see that saving a few bytes on the counter saves us a complete additional frame in many cases. While this mechanism allows us to save bandwidth, it has an impact on security. If the counter is transferred fully, the probability of the adversary to guess a MAC tag correctly is $1/2^{\ell}$ where ℓ is the length of the tag, this increases to $w_{max}/2^{\ell}$. For w_{max} we recommend a value below 4 bits. An attacker with the ability to suppress more than w_{max} messages can permanently desynchronize the sender and receiver. Since such an attacker could just suppress

all messages achieving the same effect we consider it an implementation detail to mitigate this attack.

Worst case run times. While the technique just described allows us to omit transferring parts of the message counter and reconstruct them with the MAC, this comes at the cost of additional MAC computations. To be specific, w_{max} in the worst case. Here, time constraints become a concern: There is an upper bound on the time the processing of a message may take, i.e. w_{max} MAC calculations to recover the counter must not take longer than t_{max} ms. Of course, with a sufficiently powerful receiver, a standard MAC construction could be used and either the receiver would have a sufficiently strong CPU or MAC computation could be parallelized. However, the high cost pressure in the automobile sector makes powerful receivers impossible.

Pre-compute and merge. In this paper we present MergeMAC, a dedicated MAC constructions for these constrained environments. One of the main features of MergeMAC is that it is particularly well suited for the recovery of predictable information, e.g., such as the most significant bits of the counter as described above. The idea is, basically, to split the messages in two parts, the predictable information that is not transferred and the payload part that actually is. Then a pre-MAC is computed for both parts independently (see Figure 1). For the predictable information, these pre-MACs can be pre-computed and cached - e.g. this would be the pre-MACs for the current and the $w_{max} - 1$ counter values. Two pre-MACs can then be merged into the final MAC. As the pre-MACs are already PRF outputs, we can significantly relax the security requirements for the merging function. This allows to speed this step up significantly, e.g. by roughly 70% in contrast to using a lightweight cipher for merging the PRF outputs. Our merging function Merge is a round-reduced variant of Chaskey [MMVH$^+$14]. MergeMAC has the following features:

- Our merge function Merge is derived from Chaskey and hence inherits its features: First and foremost, it is a dedicated design for microcontrollers and easy to implement in hardware.
- We define a construction for the merge function based on lightweight ciphers. Other constructions based on more secures primitives are possible but we focus on the lightweight scenario. On top, Merge is keyless allowing for very efficient hardware implementation. We further detail this in Section 2.

Contributions. Efficient merging function. We introduce the concept of a merging function and present an efficient candidate based on Chaskey. We conduct a thorough cryptanalysis for this candidate and provide a performance analysis using the SUPERCOP framework [Ber16].

Practical application. We use our merging function to define the MAC scheme MergeMAC. We analyze and compare different recommendations to instantiate MAC with a 128-bit and a 64-bit version of our merging function. Overall, we demonstrate the benefits of this construction for a practical proposal to secure communication on the CAN bus.

A security definition for merging functions. We formally define security requirements for merging functions. We show how our security definition can be applied by proving the MergeMAC to be EUF-CMA secure based on the assumption that Merge is a secure merging function and we have two secure PRFs. Consequentially, we show generically that every secure merging function can be used to instantiate MergeMAC.
Information recovery with MACs. To the best of our knowledge this paper is the first to introduce recovery of low-entropy information through a MAC to save bandwidth.

Related Work. The most popular MAC schemes today are HMAC [BCK96,Tur08] and CMAC [IK03,Dwo16]. Both cannot be parallelized and do not allow for merging messages in the way outlined above. Bellare et al. [BGR95] first discussed the possibility of parallelizing MAC calculations for increased efficiency when introducing XMAC. Later Black et al. [BR02] introduced PMAC with the express goal of providing a parallelizable MAC construction. Both approaches allow reusing partial results in a way that also allows merging of pre-computed MACs for message parts. However, merging requires an additional keyed block cipher operation which is what we avoid with our lightweight key-less merging function (see also the discussion in Section 5). In XMACR the block cipher operation is needed as the randomness is fed to the block cipher. In PMAC the final block cipher call always has to be performed, no matter if parts of the message have been reused or not. In XMACC the counter has to be fed to the block cipher. Considering that we will have several possible values for the high bits of the counter and that the low bits will be included in the message this also depends on sender input. One way to view our construction in relation to these proposals is as a specific kind of merging function which can be made more efficient than a generic constructions because the attacker has little control over the inputs.

Outline. In Section 2 we present the construction for MergeMAC and introduce the merging function Merge. In Section 3 we define the security requirements for a merging function and prove that any merging function satisfying these requirements can be used to instantiate MergeMAC. Section 4 presents the results of the cryptanalysis we conducted for our candidate of the merging function from Section 2. In Section 5 we discuss the performance of our construction before we conclude in Section 6.

2 The MergeMAC construction

In this Section, we define the MergeMAC construction in detail and give concrete parameters. MergeMAC uses two independent keys K_1, K_2 of k-bit each, to process a message msg of arbitrary size into a tag tag of n bits. These keys are used to instantiate the variable input length PRFs $\mathcal{P}_1, \mathcal{P}_2$ using any MAC scheme that is a secure PRF as outlined in Section 3, for example AES-CMAC

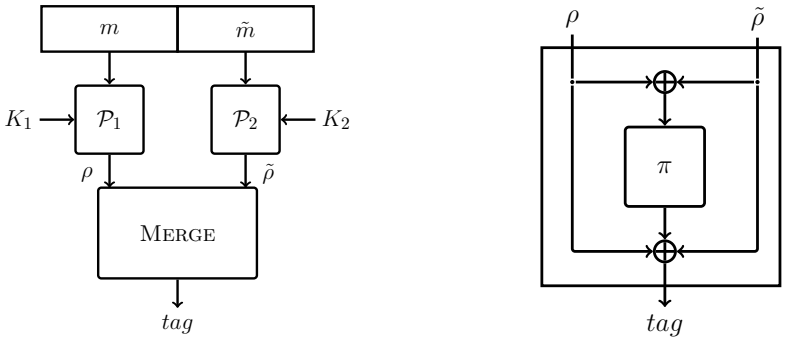

Fig. 1: (Left): MergeMAC construction, (Right): Merge function.

or Chaskey. The message msg is split into two parts, the size of both parts is flexible since we use variable input length PRFs. These parts are then combined into the MAC using a function Merge which has relaxed security requirements because it only operates on random and secret inputs. We suggest three rounds of the permutation π as defined in Section 2.3 and combine the PRF outputs by XORing them together as input to the permutation π. Furthermore, we also XOR the outputs of the PRF to the output of the permutation π. Calling the input parts $\rho, \tilde{\rho}$ this is a Davies-Meyer construction for $\rho \oplus \tilde{\rho}$. MergeMAC is illustrated in Figure 1.

2.1 Variations

Our construction allows for several design choices. It is possible to define variations of MergeMAC with different PRF instantiations or a different merging function. Since we are targeting constrained environments we suggest using lightweight functions, specifically Present or Prince in CMAC mode and a slightly adapted Chaskey round function, which will be described in more detail in Section 2.3. If further optimizations are needed for applications with higher security requirements different PRFs and merging function can be chosen. The only constraints are that any used MACs have good PRF properties and that the merging function Merge provides enough confusion to prevent an attacker from combining previous results into MACs for unknown messages. We discuss sufficient requirements for the proof in Section 3.

2.2 Caching

The main design goal when constructing MergeMAC was to provide a way to re-use calculations performed for similar messages to speed up MAC verification. This was achieved and MergeMAC allows caching the outputs of both PRFs. If a message $\mathsf{msg}' = m'||\tilde{m}$ is received the evaluation of $\tilde{\rho} = \mathcal{P}_2(m)$ can be omitted by using a cached version of $\tilde{\rho}$. This is particularly beneficial for message parts

which are constant or have low entropy for example a recipient ID. Compared to other constructions we can combine cached intermediate results without a full evaluation of any PRF. Common cache-able constructions require that at least one computationally intensive part must be repeated, usually a call to a block cipher or compression function. We improve on this by making use of the fact that the intermediate results are outputs of a PRF and a simpler computation is therefore sufficient.

2.3 The merging function *F*

Efficiently combining the outputs of the two PRFs in a secure manner is a challenging task. The natural building block for this situation would be a hash function. An ideal hash function would completely hide the PRF outputs from any adversary making them inaccessible from the top by the property of the PRFs and inaccessible from the bottom by the property of the hash function. It would also perfectly mix the two input parts preventing any attacks based on recombining individual parts of previous messages. Unfortunately hash functions or even just compression functions usually require several repetitions of a round function and are not efficient enough.
Our proposal MERGE for such a function tackles the problem with a construction similar to the Lai-Massey construction [LM91] which has been used in block ciphers like IDEA [LMM91]. The purpose of our MERGE-function is to provide enough mixing of the inputs of our MERGE-function so that it is computationally infeasible to find a pre-image of MERGE or to combine outputs for different PRFs into new MACs.
We use a modified version of the π function of CHASKEY, with a reduced word size of 16 bits to match the 64-bit block size of PRESENT or PRINCE used for the PRF. The permutation π is based on modular additions, rotations and XOR functions (ARX). The advantages of ARX-based designs are high performance, compact implementations and the possibility of constant-time implementations, preventing timing side channel attacks. The rotation constants in our permutation are chosen to be optimal for microcontrollers which often only allows efficient rotations/shifts by one or two bits, and byte-permutations by 8-bits.

3 Proof of Security

In this Section, we show that given the function MERGE satisfies easily achieved requirements and secure PRFs, our scheme is a provably secure MAC. According to Daemen and Rijmen [DR05] a MAC is considered secure if it is impossible to recover the key faster than exhaustive enumeration and impossible to forge a MAC with higher success probability than random guessing. We are not especially concerned about key recovery since it implies forgery and we want the chances of forgery to be sufficiently small to make the difference irrelevant.
We use the following experiment to define a forgery, according to Bellare et al. [BGR95].

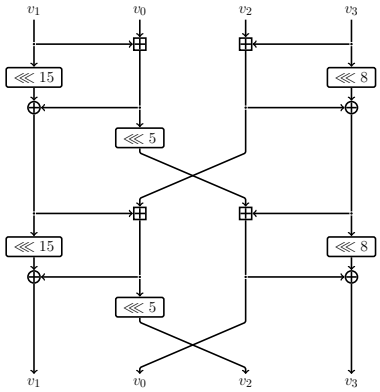

Fig. 2: One round of the permutation π that is used within our merging function MERGE.

Definition 1 (EUF-CMA**).** *The existential unforgability under chosen message experiment* EUF-CMA *consists of three algorithms*

- *The* Setup() *algorithm randomly generates a key k. It must be called first.*
- *The* MAC(msg) *algorithm will return a valid MAC for any message* msg *from the message space* $\{0,1\}^*$.
- *The verification algorithm,* Verify(msg, tag)*, will return* Valid *if* tag *is valid for* msg *and* Invalid *otherwise.*

The result of the experiment is Broken*, and the adversary wins, if a pair* (msg, tag) *verifies correctly and* MAC(msg) *has not been called, otherwise the result is* Unbroken*. An adversary which makes q_m distinct calls to* MAC*, q_v distinct calls to* Verify*, runs in time t and wins the game with probability at least ϵ is called a (t, q_m, q_v, ϵ)-adversary.*

Definition 2. *A MAC is considered (t, q_m, q_v, ϵ)-secure if there is no (t, q_m, q_v, ϵ)-adversary.*

3.1 Random Input Indistinguishability

We define a property, called Random Input Indistinguishability, which is easy to achieve in practice and which allows us to prove that our construction is a secure MAC. For simple analysis we use an unkeyed primitive similar to a hash function for the MERGE function. The usual security definitions for hash functions are collision resistance, second preimage resistance and preimage resistance. They are extensively discussed by Rogaway and Shrimpton [RS04]. Random Input Indistinguishability is even weaker than preimage resistance. Intuitively the definition is based on the following reasoning: Assuming that the PRFs are secure the only way for an adversary to learn anything about the intermediate values which are inputs to MERGE is by inverting MERGE. However finding any

preimage to $\textsc{Merge}(x, y)$ is not sufficient, instead an adversary would have to find the original x and y. Finding any other preimage is not useful since the adversary cannot find a suitable message to generate those values as long as the PRFs are secure. The only other way to mount an attack would be to find an operation that can be performed on known outputs to transform them into some other output, effectively calculating $\textsc{Merge}(\mathcal{P}_1(m), \mathcal{P}_2(\tilde{m}))$ for known m and $\tilde{m}$ but in a new combination. We formalize our requirements for Merge using the following Random Input Indistinguishability experiment RII_F:

Definition 3 (RII_F)**.** *The experiment consists of the following five algorithms*

- *The* Setup() *algorithm picks five uniformly random values: a bit b and $X, Y, U, V \in \{0,1\}^N$ where N is the bit length of our output which is the same as for the used PRFs. It must be called first.*
- *The* Query1(a) *algorithm, given any input $a \in \{0,1\}^N$ returns* $\textsc{Merge}(a, Y)$.
- *The* Query2(a) *algorithm, given any input $a \in \{0,1\}^N$ returns* $\textsc{Merge}(X, a)$.
- *The adversary can request the challenge at any time during the game. For $b = 0$* Challenge() *will return* $\textsc{Merge}(X, Y)$ *for $b = 1$ it will return* $\textsc{Merge}(U, V)$.
- *Eventually the adversary outputs a bit b' by calling* Guess(b').

The adversary wins the game if $b' = b$.

We parameterize an adversary $\mathcal{A}$ by the total amount of queries he makes q, its running time t and its probability to win ϵ. Our adversary therefore becomes a (t, q, ϵ)-adversary. Merge is (t, q, ϵ)-secure if (t, q, ϵ)-adversaries do not exist.

3.2 Reduction

In this Section, we prove the security of our scheme using two assumptions. First, the used PRFs are (t, q, ϵ_P)-secure, i.e. an adversary making q queries to the PRF and taking at most time t cannot distinguish the PRF from a random function with probability at least ϵ. Second, the function Merge is (t, q, ϵ)-secure which we will use to bound any adversary in the EUF-CMA experiment.
Our first game is $\mathcal{G}_0 = \text{EUF-CMA}$, $\mathcal{G}_1$ is the same construction but whenever $\mathcal{P}_1$ is used we replace the output with a random value. We use the notation $\mathrm{Rnd}(\cdot)$ to denominate such random oracle values and define $r_i = \mathrm{Rnd}(m_i)$ and $\tilde{r}_i = \mathrm{Rnd}(\tilde{m}_i)$ for brevity. Let $\mathcal{A}$ be a $(t, q_m, q_v, \epsilon_0)$-adversary for $\mathcal{G}_0$ and assuming that $\mathcal{P}_1$ is $(t, q_v + q_m, \epsilon_P)$-secure it follows that there must be a $(t, q_m, q_v, \epsilon_1)$-adversary for $\mathcal{G}_1$ where

$$\epsilon_0 < \epsilon_1 + \epsilon_P.$$

Game $\mathcal{G}_2$ proceeds similarly for $\mathcal{P}_2$ giving us a $(t, q_m, q_v, \epsilon_2)$-adversary such that

$$\epsilon_0 < \epsilon_2 + 2\epsilon_P.$$

We now continue with a hybrid game, successively replacing $\textsc{Merge}(r_i, \tilde{r}_i)$ with Merge evaluated on new random values. Eventually all the adversary gets is Merge evaluated on independent random values and security follows. We show an example of the initial game $\mathcal{G}_2$ in Figure 3 and give the general definition of the game $\mathcal{G}_{l+2}$ ($l \in \mathbb{N}_{>0}$) next.

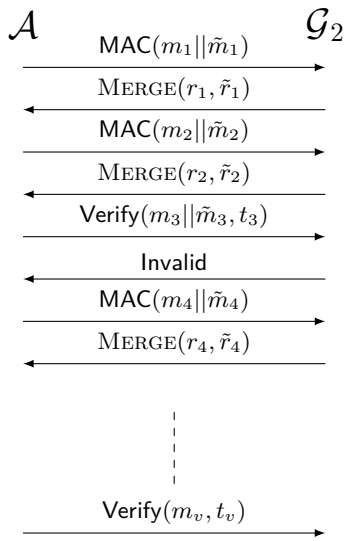

Fig. 3: **Game $\mathcal{G}_2$**

Definition 4 (Game $\mathcal{G}_{l+2}$). *The game $\mathcal{G}_{l+2}$ consists of three algorithms*

- *The* $\mathsf{Setup}()$ *algorithm randomly picks a key k. It must be called first.*
- *The result of the MAC algorithm call* $\mathsf{MAC}(m_i||\tilde{m}_i)$*, for a message* $\mathsf{msg} = m_i||\tilde{m}_i$*, depends on i:*
  - *For $i \leq l$ it will return* $\textsc{Merge}(\mathrm{Rnd}(m_i||\tilde{m}_i))$*. Here we slightly abuse notation and let* Rnd *return two values.*
  - *For $i > l$ it will return* $\textsc{Merge}(r_i, \tilde{r}_i)$ *just like $\mathcal{G}_2$.*
- *The verification algorithm,* $\mathsf{Verify}(\mathsf{msg}, \mathsf{tag})$*, will compute the MAC like* MAC *does and compare the result to* tag*.*

Going from game $\mathcal{G}_{l+1}$ to $\mathcal{G}_{l+2}$ we remove all occurrences of $\textsc{Merge}(r_l, \tilde{r}_l)$ and replace them with Merge evaluated at new random inputs $\textsc{Merge}(\mathrm{Rnd}(m_l||\tilde{m}_l))$. We now define a simulator for these two games and show that a distinguisher for $\mathcal{G}_{l+2}$ and $\mathcal{G}_{l+3}$ wins the Random Input Indistinguishability game against Merge. To simplify, we assume that the messages in queries l and $l+1$ are different. Since queries are deterministic, if a msg is repeated we simply give the same answer as the for the first query.

Definition 5 (Simulator). *The simulator for games $\mathcal{G}_{l+1}$ and $\mathcal{G}_{l+2}$ consists of three algorithms*

- *The* $\mathsf{Setup}()$ *algorithm simply calls* $\mathsf{Setup}()$ *for the Random Input Indistinguishability game.*
- *The result of the MAC algorithm* $\mathsf{MAC}(\mathsf{msg})$*, depends on i:*
  - *For $i < l$ it returns* $\textsc{Merge}(\mathrm{Rnd}(m_i||\tilde{m}_i))$

- *For $i = l$ it returns the result of a call to* Challenge() *for the Random Input Indistinguishability game.*
- *For $i > l$ the result depends on the input message* $\mathsf{msg}_i = m_i||\tilde{m}_i$*:*
  - *If $m_i = m_l, \tilde{m}_i \neq \tilde{m}_l$ it returns the result of a call to* Query1($\tilde{m}_l$) *for the Random Input Indistinguishability game.*
  - *If $m_i \neq m_l, \tilde{m}_i = \tilde{m}_l$ it returns the result of a call to* Query2(m_l) *for the Random Input Indistinguishability game.*
  - *If $m_i \neq m_l, \tilde{m}_i \neq \tilde{m}_l$ it returns* $\textsc{Merge}(r_i, \tilde{r}_i)$ *just like $\mathcal{G}_2$.*
  - *If $m_i = m_l, \tilde{m}_i = \tilde{m}_l$ this value has been queried before and the same result is used again.*

– *The verification algorithm,* Verify(msg, tag)*, will compute the MAC like* MAC *does and compare the result to* tag*.*

This implicitly maps $X = r_l, Y = \tilde{r}_l$ in the Random Input Indistinguishability game. For $i < l$ the simulator behaves like both G_{l+1} and G_{l+2}. For $i = l$ it returns Challenge() from the Random Input Indistinguishability game. If $b = 0$ this will be $\textsc{Merge}(X, Y)$ which implicitly maps to $\textsc{Merge}(r_l, \tilde{r}_l)$. For $b = 1$ it will be $\textsc{Merge}(U, V)$ which is just Merge evaluated at new random values just like $\textsc{Merge}(\text{Rnd}(m_i||\tilde{m}_i))$ in $\mathcal{G}_{l+2}$. For $i > l$ the simulator uses the query algorithms for the Random Input Indistinguishability game to uphold the implicit mapping and return $\textsc{Merge}(X, \tilde{r}_i)$ if r_i is reused or $\textsc{Merge}(r_i, Y)$ if $\tilde{r}_i$ is reused. Given a completely different input message it will behave just like $\mathcal{G}_2$ and also like both games by returning $\textsc{Merge}(r_i, \tilde{r}_i)$. If game $\mathcal{G}_{l+1}$ requires q_m, q_v queries to the simulator then at most $q_m + q_v$ queries to Merge are made. Assuming that we have a (t, q_m, q_v, ϵ_{l+1})-adversary for $\mathcal{G}_{l+1}$ and that Merge is (t, $q_m + q_v$, $\epsilon_{\textsc{Merge}}$)-secure, we have a ($t$, q_m, q_v, ϵ_{l+2})-adversary for $\mathcal{G}_{l+2}$ with

$$\epsilon_{l+1} < \epsilon_{l+2} + \epsilon_{\textsc{Merge}}.$$

If the initial game contained q_m MAC queries and q_v verify queries all the answers will have been replaced after $q_m + q_v$ steps. So for $\mathcal{G}_{q_m+q_v+2}$ the adversary receives no useful information anymore.
At this point the probability of an adversary to create a valid message-tag-pair amounts to guessing the tag. We have

$$\epsilon_{q_m+q_v+2} < 2^{-N}(q_m + q_v).$$

It follows that

$$\epsilon_0 < 2^{-N}(q_m + q_v) + \epsilon_{\textsc{Merge}}(q_m + q_v) + 2\epsilon_P.$$

This bounds all adversaries against $\mathcal{G}_0$ since we started with an arbitrary one.

4 Cryptanalysis

In this Section, we give an overview of the security related cryptographic properties of our construction. First, we discuss the security of the two initial PRFs in MergeMAC. Second, we discuss relevant attacks for the recommended merging function Merge in detail. The goal of the second analysis is to methodogically provide evidence that Merge satisfies Random Input Indistinguishability.

Table 1: Security claims according to the underlying primitives

Underlying BC	Block size	Key size	Existential forgery resistance
PRESENT	64	80	2^{-64}
PRESENT	64	128	2^{-64}
PRINCE	64	128	2^{-64}

4.1 Security of the PRFs

The security of the two initial PRFs in MERGEMAC depends on the underlying block ciphers used in CMAC mode. Table 1 gives an overview of the security claims related to the recommended MAC functions and their underlying primitives.

Birthday Attacks. While standard block ciphers, like AES have a block size of 128-bits, a majority of lightweight block ciphers have a block size of 64-bits. Bhargavan and Leurent [BL16] showed in their Sweet32 attack, that many lightweight block ciphers with 64-bit block size can be practically attacked when used in a mode like CBC. Consequently, the amount of data blocks that are processed by the initial PRFs of our construction must be limited appropriately.

Forgeries by Reordering of Messages. An adversary is capable of creating a simple forgery attack, as the first operation in the merging function MERGE is an XOR of its two inputs. The adversary just has to reorder the output tags of the PRFs as the XOR operation is commutative. Such a simple forgery attack can be prevented by using independent keys for both PRFs.

4.2 Security of the Merge Function

We chose a round-reduced and slightly adapted version of the permutation π from CHASKEY with initial and final XORs for our MERGE function. We now provide a detailed security analysis of relevant attacks on this function.

Meet-in-the-middle attacks. Meet-in-the-middle attacks (MITM) were introduced by Diffie and Hellman [DH77] in 1977 and showed that the security of double DES with two independent 56-bit keys is not as expected 112-bits, but still 56-bits. In their simplest form, MITM attacks can be described as follows: The goal is to attack a cipher E which is composed of two subciphers $E = E'_{K_1} \circ E''_{K_2}$ with keys K_1, K_2 and decryption function $D = D''_{K_2} \circ D'_{K_1}$. The adversary uses a plaintext/ciphertext pair (P, C) and computes $E''_{K_2}(P)$ under the set of all possible key values for K_2 and stores the results in a table. For the corresponding ciphertexts, the adversary computes $v = D'_{K_1}(C)$ for each value of K_1. If v is in the table, the adversary has a key candidate (K_1, K_2) which can be verified with further plaintext/ciphertext pairs.

Table 2: Differential trails for 3 of the π function.

# Rounds $\Delta_{in}(v_0, v_1, v_2, v_3) \rightarrow \Delta_{out}(v_0, v_1, v_2, v_3)$		probability
1	(0000, 0000, 8000, 0000) (8000, 0080, 0080, 8001)	1
2	(1040, 1000, 2000, 2000) (8001, 0080, 0080, 8002)	2^{-5}
3	(4010, 0010, 0020, 0020) (0201, 0A22, 0800, 0302)	2^{-14}

This has a time complexity of $2^{\#K_1} + 2^{\#K_2}$ and a memory complexity of $2^{min\{\#K_1, \#K_2\}}$ compared to exhaustive key search with a time complexity of $2^{\#K_1 + \#K_2}$ and constant memory complexity.
Applying a meet-in-the-middle attack to our construction, would require an adversary to find a pre-image of the tag by inverting the MERGE function. We show that finding a pre-image of MERGE is as hard as exhaustively guessing the internal state after the initial PRFs in Section 4.2. Moreover, as MERGEMAC does not implement an inverse function for the merging function MERGE, we also limit more advanced meet-in-the-middle attacks such as partial matching/sieve-in-the-middle attacks [BR11], splice and cut attacks [AS09] and biclique attacks [KRS12].

State recovery attacks. The goal of state recovery attacks is to recover internal state of a cipher or parts thereof. Those attacks are especially powerful against stream ciphers [MK08], as they may allow an adversary to predict the remaining key stream. Recovering the internal state in a MAC scheme could potentially lead to forgery attacks.
In MERGEMAC, recovering the internal state before the merging function MERGE would require guessing one of the PRF outputs, while keeping the other one constant. An adversary can also try to exploit the two XOR operations in the beginning and end of MERGE. However, the π function provides full diffusion after 3 rounds canceling any correlation between the input and output of π.

Differential Attacks. Differential Cryptanalysis [BS91] is one of the most powerful cryptanalytic techniques. Leurent [Leu16] applied a partitioning technique on CHASKEY, breaking 7 out of 8 rounds. We searched for differential trails of our π-function using the automated tool CryptoSMT [Ste15]. The tool is based on the constraint solver STP and the SAT solver CryptoMiniSat. Table 2 shows the results for three rounds of the π functions.

However, one can deduce that the reduced-round π-functions are not differential secure. Nevertheless, to successfully mount a differential attack on MERGEMAC an adversary needs access to differential pairs at the input and output of MERGE. Since the outputs of the PRFs are unknown an adversary does not have access to such pairs.

Internal Collisions. An internal collision is defined as any input difference Δ_{in} that maps to a zero output difference $\Delta_{out} = 0$. An adversary could use the information from internal collisions to set up a forgery attack by constructing message pairs that fulfill the input difference Δ_{in} that when colliding lead to the same tag. We experimentally verified that there are no internal collision in the merging function MERGE with our differential search tool CryptoSMT. Hence, we defined a zero difference after a selected number of rounds and checked if we can find any possible input differences that would lead to this all zero difference. We tested our π-function for up to five rounds and concluded that there are no internal collisions in our MERGE-function.

Rotational Cryptanalysis. Rotational cryptanalysis [KN10] is a generic cryptanalytic attack against ARX ciphers. Let us consider a pair $(x, x \lll r)$, consisting of plaintext x and a rotated plaintext x by r positions. Then, the pair $(x, x \lll r)$ forms a rotational pair. Rotational cryptanalysis exploits that a rotational pair remains steady through binary operations like XOR and rotation. In MERGEMAC the inputs for merging function MERGE are uniformly random and unknown thereby preventing rotational attacks.

Slide Attacks. Slide attacks were introduced by Biryukov and Wagner [BW99] in 1999 and later extended to advanced slide attacks [BW00]. Both can be used against ciphers with identical round functions. Accordingly, the number of rounds for a cipher is irrelevant against slide attacks. In our construction, each round of the permutation π in our merging function MERGE is identical and therefore vulnerable to slide attacks. However, as we XOR the inputs of the merging function MERGE before and after the permutation π, an adversary will not be able to exploit a slide attack as she would still be required to invert the XOR operation. Therefore, slide attacks do not pose a threat to our construction.

Rebound Attacks. Rebound attacks were introduced by Mendel et al. [MRST09] and are an important technique for differential cryptanalysis of hash functions. While standard rebound attacks are only applicable to AES-like compression functions Khovratovich et. al [KNR10] applied rotational cryptanalysis with rebound attacks to the SHA-3 finalist SKEIN with its THREEFISH compression function. The general idea behind the attack is to split the compression function in three parts $E = E_{fw} \circ E_{in} \circ E_{bw}$. In the inbound phase, E_{in} parts of a differential characteristic that are difficult to satisfy probabilistically are covered with a match-in-the-middle approach. In the outbound phases, the solutions from the inbound phases are propagated outwards in both directions, while it is checked if the characteristic holds. Rebound attacks are no threat to MERGE, as the adversary needs to be able to verify the differentials which is impossible because the input values are unknown. Moreover, even if an adversary uses an outbound phase over the initial PRFs, the selection of our MAC function precludes the existence of any high probability differential trails that an adversary could use to succeed.

Fixed Points. The permutation in Chaskey, which is structurally the same as our permutation π, has one fixed point. Since it only contains modular additions XOR and bit wise rotations the all zero input leads to $\pi(\mathbf{0}) = \mathbf{0}$. Fixed points can be used in a differentiability attack as shown by Maurer et al. [MRH04]. For our construction, this would mean that if an adversary can construct a message leading to an internal all zero state $\mathbf{0}$ before the application of Merge, then the tag would also be $\mathbf{0}$. However, constructing an all zero-state as input to Merge, means constructing matching PRF outputs. Since each output occurs uniformly at random an adversary will not be able to find such a plaintext with a complexity better than 2^{64}.

Algebraic Attacks. Higher-order cryptanalysis and zero-sum distinguisher were introduced by Lai in 1994 [Lai94] and first applied to block ciphers by Knudsen in 1995 [Knu95]. In 2009, Dinur and Shamir introduced Cube attacks [DS09]. All of those algebraic attacks exploit a low algebraic degree of a block cipher to attack it. In our merging function Merge, the only non-linear operation is a modular addition. This modular addition ensures that the algebraic degree grows sufficiently for each output bit. Furthermore, as each input to the merging function Merge is first processed by the underlying block cipher of the PRFs, we can ensure that the degree of the input bits are already sufficiently large (i.e. the degree is n-1, where n is the block size of the block cipher of the PRFs).

5 Performance

We benchmarked both Chaskey and our function Merge using SUPERCOP [Ber16] on an Intel i7-4600U CPU running at 2.10 GHz. While this is not a platform requiring lightweight cryptography we are only interested in comparing relative timings for which it is sufficient. We give the median of required cycles for Chaskey for several input sizes in Table 3. As is to be expected, our simplified version is faster than Chaskey on even the shortest messages since we only apply the round function three times. The median of the required cycles for Merge was 38 which, compared to the second row of Table 3, is close to 3 out of 8 rounds. The difference becomes more pronounced for longer messages. To consider the overhead of our construction let t_1, t_2 be the time required for $\mathcal{P}_1$ and $\mathcal{P}_2$ and let t_{Merge} be the time required by the merging function. So in the least favorable comparison, when the entire message would fit into one block, our construction requires time $t_1 + t_2 + t_{\text{Merge}}$ where simply using $\mathcal{P}_1$ as a MAC would only require t_1. Using our performance values for Chaskey this would be an overhead of 132%. However, since $\mathcal{P}_1$ and $\mathcal{P}_2$ can execute in parallel this immediately comes down to only 32%. For a message size of over 16 bytes, so at least two blocks, and leveraging parallel execution our construction is already as fast as the simple MAC. In the ideal case when both message parts are known and the PRF outputs are cached our construction is three times faster. Note that in the case of Chaskey the overhead required by our merging function is less

Table 3: Required cycles for hashing with Chaskey

Input size	Median cycles
0	153
16	111
32	147
64	216
128	354
256	639
512	1182
1024	2259
2048	4461
4096	8835

than the overhead required for creating the padding for messages as shown in the first line of Table 3.
For the CAN bus application this means that in the case of a correct counter our construction will at most have an overhead of 32%. In case of an incorrect counter trying additional values only costs one third of a normal MAC evaluation. In the worst case, if the counter has wrapped many times, our construction would require only one third of the time.

6 Conclusions

We presented MergeMAC, a MAC construction specifically tailored to accomplish authenticated communication in very constrained environments with limited bandwidth and the need to satisfy strict time constraints. The construction is based on a merging function for which we conducted a thorough cryptanalysis and provided performance figures. We demonstrated the applicability of MergeMAC for authenticating messages on the CAN bus. In case of different security requirements it is perfectly possible to instantiate this MAC construction with different PRFs like AES-CMAC and a suitable merging function.
One of the questions left for future work is whether there are more efficient merging functions that still meet the security requirements as defined in Section 3. The merging function Merge we defined merges two pre-MACs into the final MAC. For other use cases it is possible to canonically increase the number of pre-MACs.

References

AS09. Kazumaro Aoki and Yu Sasaki. *Preimage Attacks on One-Block MD4, 63-Step MD5 and More*, pages 103–119. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

BCK96. Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In *Crypto*, volume 96, pages 1–15. Springer, 1996.

Ber16. Daniel J. Bernstein. Supercop. `https://bench.cr.yp.to/supercop.html`, 2016.

BGR95. Mihir Bellare, Roch Guérin, and Phillip Rogaway. *XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions*, pages 15–28. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995.

BL16. Karthikeyan Bhargavan and Gaëtan Leurent. On the practical (in-)security of 64-bit block ciphers: Collision attacks on http over tls and openvpn. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 456–467, New York, NY, USA, 2016. ACM.

BR02. John Black and Phillip Rogaway. *A Block-Cipher Mode of Operation for Parallelizable Message Authentication*, pages 384–397. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

BR11. Andrey Bogdanov and Christian Rechberger. *A 3-Subset Meet-in-the-Middle Attack: Cryptanalysis of the Lightweight Block Cipher KTANTAN*, pages 229–240. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

BS91. Eli Biham and Adi Shamir. *Differential Cryptanalysis of DES-like Cryptosystems*, pages 2–21. Springer Berlin Heidelberg, Berlin, Heidelberg, 1991.

BW99. Alex Biryukov and David Wagner. *Slide Attacks*, pages 245–259. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

BW00. Alex Biryukov and David Wagner. *Advanced Slide Attacks*, pages 589–606. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

CMK+11. Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, Tadayoshi Kohno, et al. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*. San Francisco, 2011.

DH77. W. Diffie and M. E. Hellman. Special feature exhaustive cryptanalysis of the nbs data encryption standard. *Computer*, 10(6):74–84, June 1977.

DR05. Joan Daemen and Vincent Rijmen. A new mac construction alred and a specific instance alpha-mac. In *FSE*, 2005.

DS09. Itai Dinur and Adi Shamir. *Cube Attacks on Tweakable Black Box Polynomials*, pages 278–299. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

Dwo16. Morris J Dworkin. Recommendation for block cipher modes of operation: The cmac mode for authentication. *Special Publication (NIST SP)-800-38B*, 2016.

IK03. Tetsu Iwata and Kaoru Kurosawa. omac: One-key cbc mac. In *FSE*, volume 2887, pages 129–153. Springer, 2003.

ISO11. Message authentication codes (macs) – part 1: Mechanisms using a block cipher. Standard, International Organization for Standardization, Geneva, CH, March 2011.

KCR+10. Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. Experimental security analysis of a modern automobile. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 447–462. IEEE, 2010.

KN10. Dmitry Khovratovich and Ivica Nikolić. *Rotational Cryptanalysis of ARX*, pages 333–346. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

KNR10. Dmitry Khovratovich, Ivica Nikolić, and Christian Rechberger. *Rotational Rebound Attacks on Reduced Skein*, pages 1–19. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

Knu95. Lars R. Knudsen. *Truncated and higher order differentials*, pages 196–211. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995.

KRS12. Dmitry Khovratovich, Christian Rechberger, and Alexandra Savelieva. *Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family*, pages 244–263. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

Lai94. Xuejia Lai. *Higher Order Derivatives and Differential Cryptanalysis*, pages 227–233. Springer US, Boston, MA, 1994.

Lan11. Ralph Langner. Stuxnet: Dissecting a cyberwarfare weapon. *IEEE Security and Privacy*, 9(3):49–51, May 2011.

Leu16. Gaëtan Leurent. *Improved Differential-Linear Cryptanalysis of 7-Round Chaskey with Partitioning*, pages 344–371. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

LM91. Xuejias Lai and James L. Massey. *A Proposal for a New Block Encryption Standard*, pages 389–404. Springer Berlin Heidelberg, Berlin, Heidelberg, 1991.

LMM91. Xuejia Lai, James L. Massey, and Sean Murphy. *Markov Ciphers and Differential Cryptanalysis*, pages 17–38. Springer Berlin Heidelberg, Berlin, Heidelberg, 1991.

MK08. Alexander Maximov and Dmitry Khovratovich. *New State Recovery Attack on RC4*, pages 297–316. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

MMVH$^+$14. Nicky Mouha, Bart Mennink, Anthony Van Herrewege, Dai Watanabe, Bart Preneel, and Ingrid Verbauwhede. *Chaskey: An Efficient MAC Algorithm for 32-bit Microcontrollers*, pages 306–323. Springer International Publishing, Cham, 2014.

MRH04. Ueli Maurer, Renato Renner, and Clemens Holenstein. *Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology*, pages 21–39. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

MRST09. Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. *The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl*, pages 260–276. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

RS04. Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *International workshop on fast software encryption*, pages 371–388. Springer, 2004.

RSWO17. E. Ronen, A. Shamir, A. O. Weingarten, and C. O'Flynn. Iot goes nuclear: Creating a zigbee chain reaction. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 195–212, May 2017.

Ste15. Stefan Kölbl. CryptoSMT: An easy to use tool for cryptanalysis of symmetric primitives, 2015. `https://github.com/kste/cryptosmt`.

Tur08. James M Turner. The keyed-hash message authentication code (hmac). *Federal Information Processing Standards Publication*, 2008.