ChiLow and ChiChi: New Constructions for Code Encryption

Yanis Belkheyar[1], Patrick Derbez[2], Shibam Ghosh[3,6], Gregor Leander[4], Silvia Mella[1], Léo Perrin[6], Shahram Rasoolzadeh[4], Lukas Stennes[4], Siwei Sun[5,9], Gilles Van Assche[7], and Damian Vizár[8]

[1] Radboud University, Nijmegen, Netherlands
[2] Univ Rennes, Inria, CNRS, IRISA, Rennes, France
[3] University of Haifa, Haifa, Israel
[4] Ruhr University Bochum, Bochum, Germany
[5] University of Chinese Academy of Sciences, Beijing, China
[6] Inria, Paris, France
[7] STMicroelectronics, Diegem, Belgium
[8] CSEM, Neuchâtel, Switzerland
[9] State Key Laboratory of Cryptology, Beijing, China

Abstract. We study the problem of embedded code encryption, i.e., encryption for binary software code for a secure microcontroller that is stored in an insecure external memory. As every single instruction must be decrypted before it can be executed, this scenario requires an extremely low latency decryption. We present a formal treatment of embedded code encryption security definitions, propose three constructions, namely ACE1, ACE2 and ACE3, and analyze their security. Further, we present ChiLow, a family of tweakable block ciphers and a related PRF specifically designed for embedded code encryption. At the core of ChiLow, there is ChiChi, a new family of non-linear layers of *even* dimension based on the well-known χ function. Our fully unrolled hardware implementation of ChiLow, using the Nangate 15nm Open Cell Library, achieves a decryption latency of less than 280 picoseconds.

Keywords: symmetric cryptography · lightweight cryptography · memory encryption · low latency encryption

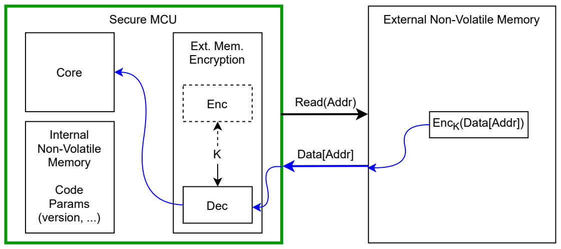
1 Introduction

In the rapidly expanding digital landscape, secure microcontrollers are a cornerstone for ensuring security and privacy across a wide range of applications, including payment systems, identification, and secure automotive solutions. These microcontrollers protect our digital assets and intellectual property continuously, defending against ever-evolving threats. Secure microcontrollers are typically composed of a core microprocessor, peripherals dedicated to specific functions such as communication or use of external memories, and on-chip memory. Although they are functionally similar to regular chips, their manufacturing is more complex due to the physical countermeasures that need to be added. To keep the production costs low, the silicon area of such secure microcontrollers should remain reasonably small, and a natural idea to achieve this is to offload the contents of memory to an external, encrypted memory. The external memory can then be a standard component that is manufactured using regular technologies.

Microcontrollers typically use such non-volatile memory to store data and to store software binary code. The former may comprise user information, the internal state of a given application, or enrollment information in the case of connected systems. It needs to be read and updated more or less regularly, depending on the application. The latter is used to store the instructions, addresses and constants that are executed by the microcontroller. Instructions are frequently fetched from it, but writes (during updates of the software) are rare.

In a similar context, low-power systems-on-chip (SoC) that integrate multiple cores, memory, and peripherals for communication, graphics, and sensor interactions on a single silicon die often rely on external memory. This is because on-chip memory alone is insufficient to store graphical resources, large sensor data, and binary code, especially as the device's features expand over its life cycle, as seen in applications like wearables.

Code Encryption Use Case We focus on the case where the software binary code has been offloaded to an external memory, outside of the secure microcontroller, as depicted below.


Secure MCU
Core
Internal Non-Volatile Memory
Code Params (version, ...)
Ext. Mem. Encryption
Enc
K
Dec
Read(Addr)
Data[Addr]
External Non-Volatile Memory
Enc_K(Data[Addr])

There is a strong need to keep the code confidential, not only to ensure the security of the system (to hinder the search for exploitable software vulnerabilities) but also to protect the intellectual properties (advanced algorithms, physical models, etc.) in the code. To protect the code from eavesdroppers, it is stored encrypted in the external memory. The microcontroller stores the secret key in an area protected from physical attacks and is then able to decrypt the instructions on the fly. In order to protect the system against active attacks inserting non-authorized code, adding integrity to the encrypted code is crucial as well.

The case of storing encrypted code in an external memory differs from the encryption of data. Mainly, it is highly unbalanced between encryption and decryption: Decryption needs to be very fast, as fast as instructions are fetched and executed, whereas the encryption is done once at the installation of the software and then only sporadically, during software updates. The latency of the decryption is therefore a critical aspect of the algorithm and its implementation, which can be assumed to be executed on a dedicated hardware circuit. In contrast, the performance of the infrequent encryption is much less critical, and it may even be acceptable that the entire software binary code destined for external memory is re-encrypted even if only a portion of it needs to be updated.

Since the secure microcontroller will decrypt instructions continuously, energy consumption of the decryption circuit is another important aspect to consider. This requirement can be simplified by aiming for an implementation that has a small area, as power and energy consumption are highly correlated to the number of gates that flip, and hence to the area.

Another aspect to discuss is granularity. The secure microcontroller must be able to randomly access instructions in the address space. Most of the times, the microprocessor needs to execute consecutive instructions, but rapid jumps are possible. To avoid buffering instructions or execution stalls, a useful choice is to opt for the encryption of 32-bit words, typically containing one or two instructions. Popular microcontrollers have a 32-bit data bus, hence the decryption inserts itself naturally between the external memory and the microprocessor.

There is also a tension between the integrity level and overhead. Clearly, one cannot hope for a strong authentication of each 32-bit instruction, as the overhead of say 128-bit tag would be unacceptable, this yielding an expansion factor of 5. Instead, a practically acceptable trade-off is to have a small expansion, in the range of 8 to 16 bits per 32-bit instruction. Of course, with such a small expansion, forgery becomes practically feasible, but this can be controlled by making forgery attempts very costly to the adversary. For example, one can assume that the secure microcontroller takes drastic measures in case it detects a forgery, such as erasing the decryption key immediately.

Finally, the adversarial resources inherent to code encryption are significantly lower than in other use cases, such as high-volume Internet-crossing secure communication. Taking an example of an embedded device with a product support lifetime of 30 years, receiving a software update no more than every month and having a generous 32MB of code allocated in the external memory, the total number of updates (each having a distinct value of a software version counter) will be no more than 360 and about 2^{32} data blocks of 32 bits processed in encryption queries in total. If the version counter is mismanaged (represented as an 8-bit integer), some of the “nonces” (composed of address-version pairs) will be reused, but only once.

All these requirements and performance constraints are essential for the significant use case of code encryption. As will be discussed, no solutions currently meet all these conditions, making it an intriguing and potentially impactful research challenge. It is this challenge we aim to solve in this work.

Prior Art Disk and memory encryption is a recurring challenge in applied cryptography. Important examples of such technologies include IBM’s SecureBlue, Intel’s SGX, and AMD’s SEV. Smartcards, like the ones of NXP, STMicroelectronics and Infineon, perform local memory encryption in an ultra-constrained setting, see, e.g., the OTFDEC scheme used on some STM32 chips [42]. Other examples include the encryption and authentication of RAM on large systems [43] and the encryption of FPGA data [45].

In many of those cases, the encryption is performed with the AES block cipher in counter mode or with AES-XTS [37,28]. This choice is however unsuited for our purposes, as it implements plain encryption without authentication or any defense against chosen ciphertext attacks, has a relatively large block size and the use of the AES block cipher does not easily yield a low-latency implementation.

On the other hand, the design of cryptographic primitives with low latency is an active line of research. Many block ciphers have been designed with this metric in mind, e.g., Aradi [24], Mantis [9], Prince [12], Princev2 [14], Qarma [4], Qarmav2 [5], and Speedy [32], to name a few. The block size of these ciphers, at least 64 bits, is however too large for our purposes, as we want to match the bus size and avoid buffering. The BipBip block cipher [10], with its 24-bit block, would be an exception. However, it does not meet our security requirements or the desired block size, making it unsuitable to us.

For authentication, low-latency pseudo-random functions (PRFs) have also been developed, see, e.g., Gleeok [1], Koala [21], Orthros [6], and `Twinkle` [44]. Gleeok and `Twinkle` have also been used to build schemes for authenticated encryption and pointer integrity. Again, these are oversized for our use case, resulting in either a latency or energy penalty or – in most cases – both.

Our Contributions We present primitives and modes to solve all the above conditions. Thus, they improve upon the state-of-the-art in several aspects.

Modeling the problem. We start by *developing a syntax and security models for code encryption in embedded systems*. Our models can be seen as a formal interpretation of existing, general AE notions for the semantics and particularities of our use case. They are a basis for both the most pertinent provable analysis of the modes as well as for an optimized design of lightweight primitives tailored to the use case and lead to three simple constructions:

ACE1 an extension of the existing, confidentiality-only solutions, such as STMicroelectronics' OTFDEC. Simple, parallelizable but not very robust.

ACE2 essentially a hardening of the above against "nonce" misuse and exploiting the redundancy in plaintext space for increased integrity.

ACE3 the most robust constructions based on encrypt-then-encipher approach, albeit less flexible than the first two, as the ciphertext expansion is not as easy to change.

We provide a security analysis of these constructions, investigating code AE security, with possibly repeated "nonces", resistance to repeated forgeries and the ability to leverage plaintext space redundancy for increased integrity in Section 2.

Primitive design. While the provable considerations build the basis for our work, we consider the primitive design to be our main contribution. We design a tweakable block cipher with a 32 or 40-bit block size, a 64-bit tweak and a 128-bit master key as well as a related PRF. Although the design adheres to a traditional SPN structure at a high level, it incorporates innovative features that enable significant improvements in both latency and energy efficiency: a nested key-schedule, and a new variant of χ.

In the well-established `TWEAKEY` framework [29] the key and the tweak are treated as (flexible) parts of one object. In most instantiations of this framework [9,30], the "tweakey" (formed by concatenating the tweak and key) is updated linearly. Instead, we design a *nested tweak-key schedule* offering a firm distinction between the tweak and the key. These separate objects are updated using non-linear operations as follows: a key schedule updates the key state to derive round keys, which are used to modify the tweak state, which in turn is used to modify the cipher state. This approach has several advantages that are well-suited for our application. First, the latency of all parts can easily be chosen to be identical, well-balancing the computational load and allowing non-linear operations also in the tweak-key schedule. Second, the later round tweaks are strong encryption of the tweak with the master key. Thus, even if an attacker managed to recover those round tweaks in an attack, it does not allow them to easily deduce information on the master key or round tweaks for other values of the tweak. Finally, this allows to precompute the round keys to reduce the overall area of a hardware implementation.

Our round functions are based on the χ function introduced already in [16]. χ represents a family of permutations for any odd number of inputs. As another main contribution, we present an infinite family of functions that maintain all the excellent properties of χ, i.e., minimal latency and good cryptographic properties, and are *permutations* when the block size is a multiple of four. While these properties could also be achieved by concatenating two smaller versions of χ over roughly half the state, our family offers the advantage of stronger diffusion and approximately twice the algebraic degree in the inverse direction compared to this straightforward approach.

Outline We start by presenting the security models for our use case in Section 2. A more detailed discussion is provided in the Appendices A and B due to the page limit. In Section 3 we define our new primitives and set the security claim. The reasoning for our choice, along with the theoretical reasoning for ChiChi is given in Section 4, again the actual proof for its bijectivity can be found in Appendix D. Section 6 is devoted to the implementation and the discussion of the performance results.

Our work opens the way for several interesting questions for future work. Those include in particular the question of how to implement our schemes in a protected manner, as well as several theoretical questions, in Section 7.

Notations We denote by $\mathbb{F}_2$ the finite field with two elements and by $\mathbb{F}_2^n$ the vector space of dimension n over $\mathbb{F}_2$. For a finite set $\mathcal{S}$ let $a \leftarrow\!\!\$\ \mathcal{S}$ denote sampling a uniformly from $\mathcal{S}$. We let $\mathbb{N}$ denote the set of all (positive) natural numbers and $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. All strings are binary strings. We denote the length of a string X as $|X|$ and denotes its i^{th} leftmost bit as X_i with $0 \leq i < |X|$. $X\|Y$ is the concatenation of strings X and Y and ε denotes the empty string of length 0. We write $\overline{X}$ for the bitwise complement. We denote the set of all strings of length n as $\{0,1\}^n$ for $n \in \mathbb{N}$ and we let $\{0,1\}^* = \bigcup_{n\in\mathbb{N}_0}\{0,1\}^n$, s.t. $\varepsilon \in \{0,1\}^*$. We let $X_{a\cdots b}$ denote the bitstring $X_a\| \ldots \|X_b$ with $0 \leq a \leq b < |X|$ and we let $X_{\cdots b}$ and $X_{a\cdots}$ be the short-hands when respectively $a = 0$ and $b = |X| - 1$. For $a > b \in \mathbb{N}$, we let $(a)_b$ denote the falling factorial $a \cdot (a-1) \cdot \ldots \cdot (a-b+1)$.

2 Authenticated Code Encryption

We adapt the nonce-based AE [41] syntax to the semantics of code encryption. An authenticated code encryption (ACE) scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ features deterministic encryption and decryption algorithms $\mathcal{E} : \mathcal{K}\times\mathcal{A}\times\mathcal{N}\times\mathcal{M} \to \mathcal{C}\cup\{\bot\}$ and $\mathcal{D} : \mathcal{K}\times\mathcal{A}\times\mathcal{N}\times\mathcal{C} \to \mathcal{M}\cup\{\bot\}$. To achieve a constant, low latency we limit ACE plaintexts to a fixed input length $\mathcal{M} = \{0,1\}^n$, primarily with $n = 32$ for 32-bit microcontrollers. Otherwise, an authenticated decryption of multiple 32-bit code fragments could stall the core, if the first decrypted fragment were a jump instruction. We consider ciphertexts $\mathcal{C} = \{0,1\}^{n+\tau}$ with a constant expansion (or stretch) $8 \leq \tau \leq 16$ to pair with $n = 32$, as higher values would lead to an unacceptable overhead in practice. The remaining arguments are *target address* $A \in \mathcal{A}$ (to retrieve the code fragment from) and an *auxiliary information* $N \in \mathcal{N}$ (such as a software version counter). Typically, $\mathcal{A} = \{0,1\}^a$ and $\mathcal{N} = \{0,1\}^\nu$, with $a = 32$ (corresponding to a 32-bit address bus) and $\nu \geq 8$. We limit ourselves to *correct* constructions, i.e., for every $(K, A, N, M) \in \mathcal{K}\times\mathcal{A}\times\mathcal{N}\times\mathcal{M}$, if $\mathcal{E}(K, A, N, M) = C$ then $\mathcal{D}(K, A, N, C) = M$.

2.1 Security Model

While the same memory address will be re-populated with different code fragments multiple times throughout a device's lifetime (through software updates), the *combined* (A, N) pair may be an effective nonce if each update has a unique version counter assigned to N. If the version counter is mismanaged, or simply not used, (A, N)-values may repeat, but the number of times any value repeats would still be limited to the number of updates. This is captured in our security notion **cae-a**. The **cae-a** advantage of an adversary $\mathscr{A}$ against Π is $\mathbf{Adv}^{\mathbf{cae\text{-}a}}_{\Pi}(\mathscr{A}) = \Pr[\mathscr{A}^{\mathbf{cae\text{-}a\text{-}re}_{\Pi}} \Rightarrow 1] - \Pr[\mathscr{A}^{\mathbf{cae\text{-}a\text{-}id}_{\Pi}} \Rightarrow 1]$ with games defined in Fig. 1. $\mathscr{A}$'s resources of interest are the number of encryption queries q_e, the number of non-trivial, adversarial decryption queries q_d, the maximal number of encryption queries done with any address-auxiliary info pair q_n, and running time t. When $q_n = 1$, then (A, N) effectively becomes a nonce. In practice, q_d can be forced to a small value, as discussed in Section 1.

When interpreted as code fragments, not all n-bit strings are valid. In an ARM microcontroller, for example, the instruction set is a strict subset of 32-bit strings. When strings outside of this subset are fetched by the core as putative instructions, addresses or data, this generates a distinguished exception in the core. A successful forgery against the encryption scheme thus need not result in execution of meaningful, unauthorized code. We capture this in a variant of **cae-a** that additionally requires a successful forgery to land in a target plaintext subspace $\mathcal{M}' \subset \{0,1\}^n$, with $\mathbf{Adv}^{\textbf{cae-a}[\mathcal{M}']}_{\Pi}(\mathscr{A}) = \Pr[\mathscr{A}^{\textbf{cae-a}[\mathcal{M}']\textbf{-re}_{\Pi}} \Rightarrow 1] - \Pr[\mathscr{A}^{\textbf{cae-a}[\mathcal{M}']\textbf{-id}_{\Pi}} \Rightarrow 1]$, using the games from Fig. 1. The target plaintext set is a parameter of the notion, as the $\mathcal{M}'$ differs between MCUs (with a different instruction set, address layout etc.). This is also useful for evaluating whether an ACE scheme leverages code redundancy to enhance overall integrity, as well as for assessing the likelihood of forgery targeting specific instructions by considering a smaller $\mathcal{M}'$. For example, altering a conditional jump in an authentication check to a no-op could render that check ineffective.

Through the lens of the **cae-a** notions, it is impossible to separate fragile and robust constructions, as schemes that lose all security after the first successful forgery and those that don't may have a similar **cae-a** bound, for example. For this, we propose *robust* ACE notion **cae-r**, modelled after Robust AE by Hoang et. al. [26]. It is based on an all-in-one indistiguishability experiment with a *random, τ-expanding injection.* The adversarial advantage $\mathbf{Adv}^{\textbf{cae-r}}_{\Pi}(\mathscr{A}) = \Pr[\mathscr{A}^{\textbf{cae-r-re}_{\Pi}} \Rightarrow 1] - \Pr[\mathscr{A}^{\textbf{cae-r-id}_{\Pi}} \Rightarrow 1]$ is defined with games in Fig. 1. A

```
proc initialize                         cae-a $\boxed{[\mathcal{M}']}$-re$_\Pi$
$K \leftarrow\$ \mathcal{K}$
oracle Enc$(A, N, M)$
$\boxed{\textbf{if } M \notin \mathcal{M}' \textbf{ then return } \perp}$
$C \leftarrow \mathcal{E}(K, A, N, M)$
return $C$
oracle Dec$(A, N, C)$
$M \leftarrow \mathcal{D}(K, A, N, C)$
$\boxed{\textbf{if } M \notin \mathcal{M}' \textbf{ then return } \perp}$
return $M$
```

```
proc initialize                         cae-a $\boxed{[\mathcal{M}']}$-id$_\Pi$
oracle Enc$(A, N, M)$
$\boxed{\textbf{if } M \notin \mathcal{M}' \textbf{ then return } \perp}$
$C \leftarrow\$ \{0,1\}^{m+\tau}$
return $C$
oracle Dec$(A, N, C)$
return $\perp$
```

```
proc initialize                         cae-r-re$_\Pi$
$K \leftarrow\$ \mathcal{K}$
oracle Enc$(A, N, M)$
return $\mathcal{E}(K, A, N, M)$
oracle Dec$(A, N, C)$
$M \leftarrow \mathcal{D}(K, A, N, C)$
return $M$
```

```
proc initialize                         cae-r-id$_\Pi$
$\forall A \in \mathcal{A}, N \in \mathcal{N} : F_{A,N} \leftarrow\$ \mathrm{Inj}(n, n+\tau)$
oracle Enc$(A, N, M)$
return $F_{A,N}(M)$
oracle Dec$(A, N, C)$
$M \leftarrow F^{-1}_{A,N}(C)$
return $M$
```

Fig. 1: Security games for the **cae-a** (boxed lines omitted), **cae-r** and **cae-a**$[\mathcal{M}']$ (boxed lines included). $\mathrm{Inj}(n, n+\tau)$ is the set of all injective functions from $\{0,1\}^n$ to $\{0,1\}^{n+\tau}$. W.l.o.g., the adversary is assumed not to make trivial queries, e.g., a decryption of (A, N, C) with a C from an encryption of (A, N, M).

cae-r security implies aspirational ACE security, including with targeted forgery resistance (Theorem 1) and resistance to reforgeries (Lemma 1).

Finally, ACE schemes that are not **cae-r** secure may still not fully break after the first forgery. Here, we lean on the work of Forler et. al. [22], who studied *reforgeries*, requiring that forging j ciphertexts requires roughly j times the resources (computation and oracle queries) as forging once.

Theorem 1. *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an ACE scheme with $\mathcal{M} = \{0,1\}^n$ for some $n \in \mathbb{N}$ and $\mathcal{M}' \subset \mathcal{M}$ be a target plaintext set. We have for any adversary $\mathscr{A}$ with resources as defined above that*

$$\mathbf{Adv}_{\Pi}^{\textbf{cae-a}}(\mathscr{A}) \leq \mathbf{Adv}_{\Pi}^{\textbf{cae-r}}(\mathscr{A}) + 2q_d/2^\tau + q_e(q_n - 1)/2^{n+\tau}.$$
$$\mathbf{Adv}_{\Pi}^{\textbf{cae-a}[\mathcal{M}']}(\mathscr{A}) \leq \mathbf{Adv}_{\Pi}^{\textbf{cae-r}}(\mathscr{A}) + 2q_d|\mathcal{M}'|/2^{n+\tau} + q_e(q_n - 1)/2^{n+\tau}.$$

if $(q_d + q_n) \leq 2^{\tau-1}$ and, for the second statement, if $\mathscr{A}$ asks no encryption queries from $\mathcal{M}\backslash\mathcal{M}'$.

Theorem 1 is a corollary of Theorem 1 by Hoang et. al. [26], as **cae-r** and **cae-a** are respectively variants of PRI and MRAE notions for a primitive with a slightly different signature from AEAD; the difference in the second term is due to a different way of characterizing adversarial resources. For the second bound, the forgery probability is additionally multiplied by $|M'|/2^n$, accounting for the need to hit the target plaintext set.

In Lemma 1, we show that whenever there are ℓ preimage-image pairs known and q non-image elements of the domain known for a random injection, whether learned through forward or backward queries, finding another image through backward queries takes about $2^\tau/2$ attempts, as long as the total number of all queries is below $2^{n+\tau-1}$.

Lemma 1. *Let $f \leftarrow\!\!\$\ \mathrm{Inj}(n, n+\tau)$. For any $M_1, \ldots, M_\ell, C_1, \ldots, C_q$ such that $\forall 1 \leq i \leq q : f^{-1}(C_i) = \bot$ and any $C'_1, \ldots, C'_{q_d} \notin \{f(M_1), \ldots, f(M_\ell), C_1, \ldots C_q\}$. Let forge be the event that $\exists 1 \leq i \leq q_d$ such that $f^{-1}(C'_i) \neq \bot$. We have*

$$\Pr[\mathsf{forge} | f(M_1), \ldots, f(M_\ell), C_1, \ldots C_q] \leq (2q_d)/(2^\tau)$$

if $(q + \ell + q_d) \leq 2^{n+\tau-1}$.

2.2 Auxiliary Definitions

Before diving into the ACE constructions, we give the necessary definitions. We measure the (in)security of a construction Π w.r.t. a security property **xxx** using the resource parameterized function

$$\mathbf{Adv}_{\Pi}^{\textbf{xxx}}(\mathbf{r}) = \max_{\mathscr{A}}\{\mathbf{Adv}_{\Pi}^{\textbf{xxx}}(\mathscr{A})\},$$

where the maximum is taken over all adversaries $\mathscr{A}$ which use resources bounded by $\mathbf{r}$.

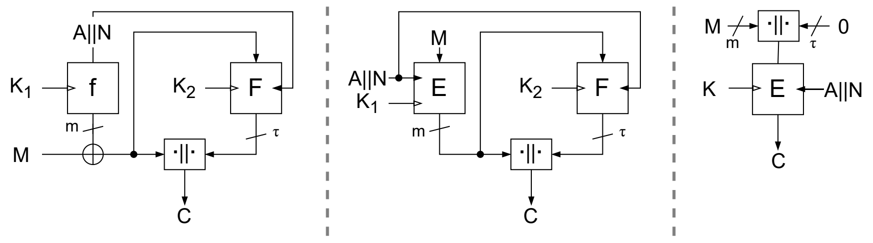


Fig. 2: ACE constructions ACE1, ACE2, ACE3 (from left to right). Here, f is a stream cipher, F is a "tweakable" PRF and E is a tweakable block cipher.

Let $\mathrm{Perm}(n)$ be the set of all permutations over n-bit strings. Let $\mathrm{Perm}^{\mathcal{T}}(n) \subseteq \{\widetilde{\pi} : \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n\}$ be the set of all tweakable permutations, i.e. for every $\widetilde{\pi} \in \mathrm{Perm}^{\mathcal{T}}(n)$, $\widetilde{\pi}(t,\cdot)$ is a permutation for every $t \in \mathcal{T}$. $\mathcal{T}$ is the set of tweaks. We use $\widetilde{\pi}^t(\cdot)$ and $\widetilde{\pi}(t,\cdot)$ interchangeably. Let $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ be a block cipher and let $\widetilde{E} : \mathcal{K} \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$ be a tweakable block cipher with a non-empty, finite $\mathcal{K} \subseteq \{0,1\}^*$. Let D and $\widetilde{D}$ denote the inverses of E and $\widetilde{E}$ respectively. Let $E_K(\cdot) = E(K,\cdot)$ and $\widetilde{E}_K^t(\cdot) = \widetilde{E}(K,t,\cdot)$. Let $\mathscr{A}$ be an adversary. Then:

$$\mathbf{Adv}_E^{\pm\mathrm{prp}}(\mathscr{A}) = \Pr\left[K \leftarrow\$ \mathcal{K} : \mathscr{A}^{E_K,D_K} \Rightarrow 1\right] - \Pr\left[\pi \leftarrow\$ \mathrm{Perm}(n) : \mathscr{A}^{\pi,\pi^{-1}} \Rightarrow 1\right]$$
$$\mathbf{Adv}_{\widetilde{E}}^{\pm\widetilde{\mathrm{prp}}}(\mathscr{A}) = \Pr\left[K \leftarrow\$ \mathcal{K} : \mathscr{A}^{\widetilde{E}_K,\widetilde{D}_K} \Rightarrow 1\right] - \Pr\left[\widetilde{\pi} \leftarrow\$ \mathrm{Perm}^{\mathcal{T}}(n) : \mathscr{A}^{\widetilde{\pi},\widetilde{\pi}^{-1}} \Rightarrow 1\right] .$$

The adversarial resources of interest are $\mathscr{A}$'s time complexity (t), total number of queries (q), and the maximal number of queries asked per any tweak (q_t).

Let $\mathrm{Func}(\mathcal{X}, n)$ be the set of all functions from $\mathcal{X}$ to $\{0,1\}^n$ for $n \in \mathbb{N}$. Let $F : \mathcal{K} \times \mathcal{X} \to \{0,1\}^n$ be a keyed function with a non-empty, finite $\mathcal{K} \subseteq \{0,1\}^*$. Let $\mathscr{A}$ be an adversary. Then, $\mathbf{Adv}_F^{\mathrm{prf}}(\mathscr{A}) = \Pr\left[K \leftarrow\$ \mathcal{K} : \mathscr{A}^{F_K} \Rightarrow 1\right] - \Pr\left[f \leftarrow\$ \mathrm{Func}(\mathcal{X}, n) : \mathscr{A}^f \Rightarrow 1\right]$. The adversarial resources of interest are $\mathscr{A}$'s time complexity (t) and the total number of queries (q).

2.3 Constructions

We investigate three simple and natural ACE constructions tailored for low latency code execution in 32-bit microcontrollers. Their encryption algorithms are depicted in Fig. 2 and their decryption algorithms follow naturally, relying on the verification of an authentication tag, or on asserting that there is expected redundancy for integrity check.

ACE1. Our first construction ACE1 can be seen as a natural extension of the existing confidentiality-only approaches, with an authentication tag simply added on top of a stream cipher encryption. While ACE1 is **cae-a**-secure (when $q_n = 1$) and resistant to reforgeries (similarly as ACE2), it is broken as soon as $q_n > 1$

and it also succumbs to targeted forgeries. Especially the latter weakness can be detrimental to secure embedded systems. Given a ciphertext tuple (A, N, C) corresponding to a known plaintext M, an attacker can *deterministically* force the MCU to execute an arbitrary fragment M' by an exhaustive search for the tag for the core ciphertext $C_{0\cdots n-1} \oplus M \oplus M'$, reusing (A, N). With a short tag, say $\tau = 8$, an attacker may abuse this to bypass an authentication check (by replacing a jump instruction with a no-op), for example. Even if the MCU drops the memory decryption key after the first detected forgery, the attacker may simply attack ≈ 256 devices until succeeding, for example to unlock memory protection and extract the device software. We therefore recommend to focus on the next two constructions.

ACE2. Our second construction, ACE2, is a hardened variant of ACE1, directly applying an n-bit TBC E to the plaintext instead of the XORing of a keystream, lending it resistance to attacks with $q_n > 1$ (Theorem 2). ACE2 is parameterized with a TBC $\mathsf{E} : \mathcal{K} \times (\mathcal{A} \times \mathcal{N}) \times \{0,1\}^n \to \{0,1\}^n$ and a PRF $\mathsf{F} : \mathcal{K} \times \mathcal{A} \times \mathcal{N} \times \{0,1\}^n \to \{0,1\}^\tau$. Similarly, as ACE1, it has a parallelizable decryption and has an easily adjustable stretch, by adopting F with a sufficiently long output and truncating it as desired. ACE2 may be instantiated with a single primitive (TBC), using a single secret key, if an appropriate domain separation is ensured in the tweak. In ChiLow-$(32 + \tau)$, this is done implicitly, where a "silent" tweak component effectively acts as a selector of disjoint tweak state subsets used between the two calls. ACE2 is secure against reforgeries, being an instance of Forler et al.'s "*Independence of F_{IV} and F_T*"-paradigm [22]. In addition, it also resists targeted forgeries (Theorem 2). Noting that the quantitative degradation terms vanish when $q_n = 1$, ACE2 makes for a pragmatic, well-rounded construction.

Theorem 2. *Let $\mathsf{E} : \mathcal{K}_1 \times (\mathcal{A} \times \mathcal{N}) \times \{0,1\}^n \to \{0,1\}^n$ be a tweakable block cipher with tweak space $\mathcal{A} \times \mathcal{N}$ and $\mathsf{F} : \mathcal{K}_2 \times \mathcal{A} \times \mathcal{N} \times \{0,1\}^n \to \{0,1\}^\tau$ be a PRF. Let $\mathcal{M}' \subset \{0,1\}^n$ be a target plaintext set. Then, for $\Pi = ACE2[\mathsf{E}, \mathsf{F}]$, we have*

$$\begin{aligned}
\mathbf{Adv}_{\Pi}^{\textbf{cae-a}}(q_e, q_d, q_n, t) &\leq \mathbf{Adv}_{\mathsf{E}}^{\pm\widetilde{\mathrm{prp}}}(q_e + q_d, t') + \mathbf{Adv}_{\mathsf{F}}^{prf}(q_e + q_d, t'') \\
&\quad + \frac{q_e(q_n - 1)}{2^n} + \frac{q_d}{2^\tau} \\
\mathbf{Adv}_{\Pi}^{\textbf{cae-a}[\mathcal{M}']}(q_e, q_d, q_n, t) &\leq \mathbf{Adv}_{\mathsf{E}}^{\pm\widetilde{\mathrm{prp}}}(q_e + q_d, t') + \mathbf{Adv}_{\mathsf{F}}^{prf}(q_e + q_d, t'') \\
&\quad + \frac{q_e(q_n - 1)}{2^n} + \frac{2q_d|\mathcal{M}'|}{2^{n+\tau}}
\end{aligned}$$

where $t' = \alpha(q_e + q_d)$ and $t'' = \beta(q_e + q_d)$ where α and β are constants dependent on the model of computation.

The proof of Theorem 2 is a standard hybrid argument, followed by an RP-RF switch [25] for each of E's used tweaks and a bound on tag guessing. The analysis of the second bound proceeds similarly, except the order of RP-RF switch and bounding of forgery probability is reversed, so that the additional constraint of forgery falling in $\mathcal{M}'$ can be evaluated using ACE2's n-bit tweakable permutation.

ACE3. Our third construction, ACE3, is a fixed-input length incarnation of the encode-then-encipher [11] paradigm, where the plaintext block is padded with τ zeros and then enciphered. It is parameterized by a TBC $\mathsf{E} : \mathcal{K} \times (\mathcal{A} \times \mathcal{N}) \times \{0,1\}^{n+\tau} \rightarrow \{0,1\}^{n+\tau}$. ACE3 achieves the robust **cae-r** security, formalized in Theorem 3; as long as the underlying TBC is unbroken, ACE3 will behave as a random injection. ACE3 has a potential for implementation with low energy consumption, as it only requires a single cryptographic accelerator with a slightly wider datapath compared to two parallel primitives in ACE2. On the other hand, ACE3 is less flexible: once E is fixed, the stretch τ can no longer be changed.

Theorem 3 ([27]). *Let $\mathsf{E} : \mathcal{K} \times (\mathcal{A} \times \mathcal{N}) \times \{0,1\}^{n+\tau} \rightarrow \{0,1\}^{n}$ be a tweakable block cipher with tweak space $\mathcal{A} \times \mathcal{N}$. Then, for $\Pi = ACE3[\mathsf{E}, \tau]$, we have*

$$\mathbf{Adv}_{\Pi}^{\textbf{cae-r}}(q_e, q_d, q_n, t) \leq \mathbf{Adv}_{\mathsf{E}}^{\pm\widetilde{\mathrm{prp}}}(q_e, q_d, q_n, t')$$

where $t' = \alpha(q_e + q_d)$ with α a constant dependent on the model of computation.

3 Primitives

In this section, we set out to define primitives needed to instantiate the ACE2 and ACE3 constructions. The former instance is called ChiLow-$(32+\tau)$ and the latter ChiLow-40. For ChiLow-$(32+\tau)$, we define two cryptographic objects. Unlike the ACE2 construction, we use the same key for the two objects, although we make them act independently by absorbing different parts of the tweak state.

First, we need a tweakable block cipher $Y = D_{32}(K, T, X)$ that takes as input a 128-bit key K, a 64-bit tweak T and a 32-bit input block X and that returns a 32-bit output block Y. In the context of ACE2, D_{32} corresponds to E^{-1} and is used to decrypt an encrypted instruction of 32 bits, hence X represents a ciphertext while Y typically represents a plaintext.

Then, we need a pseudo-random function $Z = \mathsf{F}(K, T, X)$ that takes the same inputs as D_{32} and returns a short output Z of $\tau = 8$ or 16 bits. We define another tweakable block cipher D'_{32} and define $\mathsf{F}(K, T, X)$ as the output of $D'_{32}(K, T, X)$ truncated to its first τ bits. The use for F is to compute an authentication tag for an encrypted instruction of 32 bits.

For ChiLow-40, we define a single object: a tweakable block cipher $Y = D_{40}(K, T, X)$ that takes as input a 128-bit key K, a 64-bit tweak T and a 40-bit input block X and that returns a 40-bit output block Y. In the context of ACE3, the inverse block cipher D_{40}^{-1} corresponds to E and is used to encrypt a 32-bit instruction followed by 8 bits equal to zero. The block cipher D_{40} is then used to decrypt the ciphertext, and for authentication, the caller needs to check that the last 8 bits are equal to zero.

As the overall approach to building these three tweakable block ciphers, we divide the internal state into three distinct parts and allow tweak and key material to flow from one to the next.

1. The *key state* K, which is 128-bit long, and whose content depends only on the 128-bit master key K. It is updated using the 128-bit permutations $\mathcal{K}_i$, where i is simply the round index.
2. The *tweak state* T, which is 64-bit long, that is initialized with the tweak T and whose content is updated using a family of 64-bit permutations $\mathcal{T}_k$ indexed by a 64-bit value.
3. The *cipher state* X (or X'), which is 32-bit long for D_{32} and D'_{32} or 40-bit long for D_{40}, that is initialized with the input block X. The cipher state is updated using a 32-bit (or 40-bit) round function $\mathcal{R}_t$ that uses 32 bits (or 40 bits) of the tweak state. In the context of the ACE2 construction, we actually have two cipher states, one for D_{32} and one for D'_{32}, and each is updated with a different part of the tweak state.

3.1 Key Schedule

The key schedule uses a 128-bit state $\mathsf{K}^{(i)}$ that is initialized with the master key, so that $\mathsf{K}^{(0)} = K$. Then, the state is updated with the round function $\mathcal{K}[i]$ that depends on the round number, that is, $\mathsf{K}^{(i+1)} = \mathcal{K}[i](\mathsf{K}^{(i)})$.

The round function $\mathcal{K}[i]$ is composed of three steps: the addition of a round constant $c^{(i)}$, the non-linear mapping χ_{128} and the linear mapping L_{128}:

$$\mathsf{K}^{(i+1)} = \mathcal{K}[i](\mathsf{K}^{(i)}) = L_{128}(\chi_{128}(\mathsf{K}^{(i)} \oplus c^{(i)})).$$

The round constants $c^{(i)}$ potentially affect bits 96 till 127 and are defined in Eq. (2). We give the linear and non-linear mappings in Sections 3.5 and 3.6.

3.2 Tweak Schedule

The tweak schedule uses a 64-bit state $\mathsf{T}^{(i)}$ that is initialized with the input tweak and whitened with key material, namely, $\mathsf{T}^{(0)} = T \oplus K_{0\ldots63}$. Then, the state is updated with the round function $\mathcal{T}[\mathsf{K}^{(i+1)}]$ that depends on the key schedule state. Specifically, we update the state as $\mathsf{T}^{(i+1)} = \mathcal{T}[\mathsf{K}^{(i+1)}](\mathsf{T}^{(i)})$.

The round function is composed of three steps: the non-linear mapping χ_{64}, the linear mapping L_{64} and the addition of the round key. The first 64 bits of the key schedule state are added bitwise to the tweak schedule state, giving

$$\mathsf{T}^{(i+1)} = \mathcal{T}[\mathsf{K}^{(i+1)}](\mathsf{T}^{(i)}) = L_{64}(\chi_{64}(\mathsf{T}^{(i)})) \oplus \mathsf{K}^{(i+1)}_{0\ldots63}.$$

The last round is composed only of the linear mapping, namely,

$$\mathsf{T}^{(\mathsf{rnd})} = \mathcal{T}(\mathsf{T}^{(\mathsf{rnd}-1)}) = L_{64}(\mathsf{T}^{(\mathsf{rnd}-1)}).$$

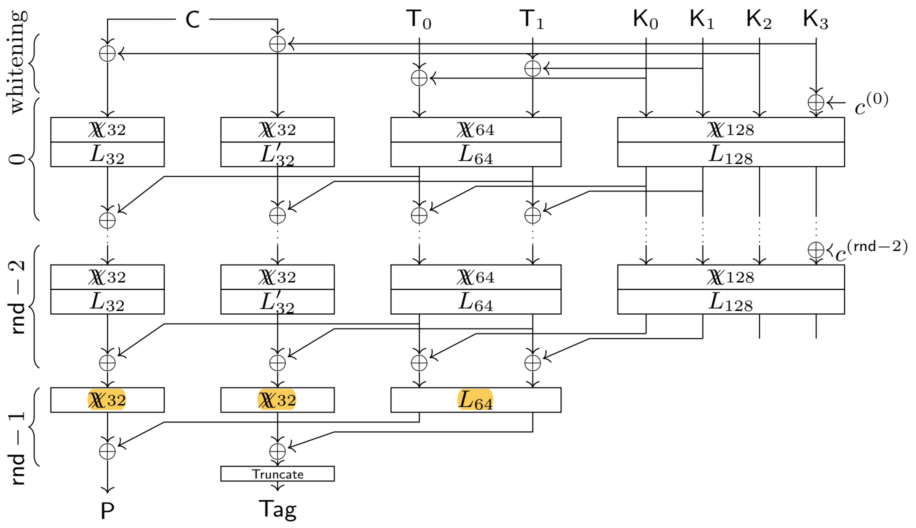


Fig. 3: A full ChiLow-(32+τ) decryption and tag computation. Wires represent 32-bit values. Here, K_i denotes the bits $\mathsf{K}_{32i\ldots32i+31}$, and similarly for T.

3.3 Data Path in ChiLow-(32 + τ)

In the ACE2 construction, the data path uses a 32-bit state $\mathsf{X}^{(i)}$ that is initialized with the input block and whitened with key material, namely, $\mathsf{X}^{(0)} = X \oplus K_{64\ldots95}$ (for D_{32}) and $\mathsf{X}'^{(0)} = X \oplus K_{96\ldots127}$ (for D'_{32}). Then, the state is updated with the round function $\mathcal{R}[\mathsf{T}^{(i+1)}]$ (for D_{32}) or $\mathcal{R}'[\mathsf{T}^{(i+1)}]$ (for D'_{32}) that depends on the tweak state. For the last round, only the non-linear mapping is applied. In more detail, the round function for D_{32} is as follows:

$$\begin{aligned} \mathsf{X}^{(i+1)} &= \mathcal{R}[\mathsf{T}^{(i+1)}](\mathsf{X}^{(i)}) &&= L_{32}(\chi_{32}(\mathsf{X}^{(i)})) \oplus \mathsf{T}^{(i+1)}_{0\ldots31} \quad \text{for } i \le \mathsf{rnd}-2, \\ \mathsf{X}^{(\mathsf{rnd})} &= \mathcal{R}[\mathsf{T}^{(\mathsf{rnd})}](\mathsf{X}^{(\mathsf{rnd}-1)}) &&= \chi_{32}(\mathsf{X}^{(\mathsf{rnd}-1)}) \oplus \mathsf{T}^{(\mathsf{rnd})}_{0\ldots31}. \end{aligned}$$

The output block Y is $\mathsf{X}^{(\mathsf{rnd})}$.

The round function of D'_{32} is very similar to that of D_{32}, but it uses a different linear mapping and takes a different part of the tweak state:

$$\begin{aligned} \mathsf{X}'^{(i+1)} &= \mathcal{R}'[\mathsf{T}^{(i+1)}](\mathsf{X}'^{(i)}) &&= L'_{32}(\chi_{32}(\mathsf{X}'^{(i)})) \oplus \mathsf{T}^{(i+1)}_{32\ldots63} \quad \text{for } i \le \mathsf{rnd}-2, \\ \mathsf{X}'^{(\mathsf{rnd})} &= \mathcal{R}'[\mathsf{T}^{(\mathsf{rnd})}](\mathsf{X}'^{(\mathsf{rnd}-1)}) &&= \chi_{32}(\mathsf{X}'^{(\mathsf{rnd}-1)}) \oplus \mathsf{T}^{(\mathsf{rnd})}_{32\ldots63}. \end{aligned}$$

The output Z is obtained by truncating $\mathsf{X}'^{(\mathsf{rnd})}$ to the first τ bits, $Z = \mathsf{X}'^{(\mathsf{rnd})}_{0\ldots\tau-1}$. We depict this in Fig. 3. We also give an algorithmic description in Algorithm 1.

3.4 Data Path in ChiLow-40

In ChiLow-40, the data path uses a 40-bit state $\mathsf{X}^{(i)}$ and 40-bit mappings. It is initialized with the input block and whitened with key material, namely, $\mathsf{X}^{(0)} = X \oplus K_{64\ldots103}$. For the rest, D_{40} is very similar to D_{32} or D'_{32}:

$$\begin{aligned} \mathsf{X}^{(i+1)} &= \mathcal{R}[\mathsf{T}^{(i+1)}](\mathsf{X}^{(i)}) &&= L_{40}(\chi_{40}(\mathsf{X}^{(i)})) \oplus \mathsf{T}^{(i+1)}_{0\ldots39} \quad \text{for } i \leq \mathsf{rnd}-2, \\ \mathsf{X}^{(\mathsf{rnd})} &= \mathcal{R}[\mathsf{T}^{(\mathsf{rnd})}](\mathsf{X}^{(\mathsf{rnd}-1)}) &&= \chi_{40}(\mathsf{X}^{(\mathsf{rnd}-1)}) \oplus \mathsf{T}^{(\mathsf{rnd})}_{0\ldots39}. \end{aligned}$$

The output block Y is $\mathsf{X}^{(\mathsf{rnd})}$ and is expected to contain the 32-bit instruction followed by 8 bits equal to zero.

For completeness, we also give an algorithmic description in Algorithm 2.

3.5 Low-Latency Linear Mappings

The linear mappings L_{128}, L_{64}, L_{40}, L_{32} and L'_{32} all have the same structure. Let $y = L(x)$ with $x, y \in \mathbb{F}_2^{128 \text{ or } 64 \text{ or } 40 \text{ or } 32}$. Then,

$$y_i = x_{\alpha i + \beta_0} + x_{\alpha i + \beta_1} + x_{\alpha i + \beta_2},$$

where the indices must be computed modulo the vector size. The values of α and β_js are given in Table 1.

3.6 Low-Latency Non-Linear Mapping

For m even and $n = 2m$, we define $\chi_n : \mathbb{F}_2^n \to \mathbb{F}_2^n, x \mapsto y$ by

$$y_i = \begin{cases} x_i + \overline{x}_{i+1} x_{i+2} & i < m-3 \text{ or } m < i < n-2 \\ x_m + \overline{x}_{m-2} x_0 & i = m-3 \\ x_{m-1} + \overline{x}_0 x_1 & i = m-2 \\ \overline{x}_{m-3} + \overline{x}_m \overline{x}_{m+1} & i = m-1 \\ x_{m-2} + \overline{x}_{m+1} x_{m+2} & i = m \\ x_{n-2} + \overline{x}_{n-1} x_{m-1} & i = n-2 \\ x_{n-1} + \overline{x}_{m-1} x_m & i = n-1. \end{cases} \tag{1}$$

Table 1: Offsets for the linear maps.

Linear Map	α	β_0	β_1	β_2
L_{32}	11	5	9	12
L'_{32}	11	1	26	30
L_{40}	17	1	9	30
L_{64}	3	1	26	50
L_{128}	17	7	11	14

3.7 Security Claim

For ChiLow-$(32+\tau)$, we claim the tweakable strong pseudo-random permutation ($\pm\widetilde{\text{prp}}$) security of the underlying block cipher D_{32} and the pseudo-random function (prf) security of D'_{32} truncated to τ bits (denoted $\lfloor D'_{32} \rfloor_\tau$). Furthermore, since these two functions are used with the same key in our ACE instances, we claim their joint security in this scenario. Our claims hold as long as the number of queries with the same tweak (q_t) and the total number of queries (q) respect the limits below.

Claim 1 *If keyed with a uniformly distributed secret key of* 128 *bits, and if* $q_t \leq 2^8$ *and* $q \leq 2^{40}$, *we claim that* D_{32} *and* D'_{32} *satisfy*

$$\begin{aligned}
\mathbf{Adv}^{\pm\widetilde{\text{prp}}}_{D_{32}}(t, q, q_t) &\leq \frac{t}{2^{128}} \\
\mathbf{Adv}^{\pm\text{prf}}_{\lfloor D'_{32} \rfloor_\tau}(t, q, q_t) &\leq \frac{t}{2^{128}} + f_{\text{Stam}}(n, \tau, q_t) \\
\mathbf{Adv}^{\pm\widetilde{\text{prp}},\text{prf}}_{D_{32}, \lfloor D'_{32} \rfloor_\tau}(t, q, q_t) &\leq \frac{t}{2^{128}} + f_{\text{Stam}}(n, \tau, q_t),
\end{aligned}$$

where the adversarial resources are the time complexity (t), *the total number of queries* (q), *and the maximal number of queries asked per any tweak* (q_t), *where*

$$\begin{aligned}
\mathbf{Adv}^{\pm\widetilde{\text{prp}},\text{prf}}_{\widetilde{E},F}(\mathscr{A}) = \Pr&\left[K \leftarrow\!\!\$\ \mathcal{K} : \mathscr{A}^{\widetilde{E}_K, \widetilde{E}_K^{-1}, F_K} \Rightarrow 1\right] \\
&- \Pr\left[\widetilde{\pi} \leftarrow\!\!\$\ \mathit{Perm}^{\mathcal{T}}(n), f \leftarrow\!\!\$\ \text{Func}(\mathcal{T} \times \{0,1\}^n, \tau) : \mathscr{A}^{\widetilde{\pi}, \widetilde{\pi}^{-1}, f} \Rightarrow 1\right]
\end{aligned}$$

and where

$$f_{\text{Stam}}(n, \tau, q_t) = \frac{1}{2}\sqrt{\frac{(2^\tau - 1)q_t(q_t - 1)}{(2^n - 1)(2^n - q_t + 1)}}.$$

Here, $f_{\text{Stam}}(n, \tau, q_t)$ gives an upper bound on the advantage of distinguishing, in q_t queries, an n-bit permutation truncated to τ bits from a random function. The formula is due to Stam [23] and satisfies $f_{\text{Stam}}(n, \tau, q_t) = 0$ if $q_t \leq 1$ and $f_{\text{Stam}}(n, \tau, q_t) \leq q_t / 2^{n-\tau/2}$ if $q_t \leq \frac{3}{4}2^n$.

Under Claim 1 and Theorem 2, ChiLow-$(32+\tau)$ is an instantiation of ACE2 achieving **cae-a** security up to the following advantage

$$\mathbf{Adv}^{\textbf{cae-a}}_{\text{ChiLow-}(32+\tau)}(q_e, q_d, q_n, t) \leq \frac{t}{2^{128}} + f_{\text{Stam}}(32, \tau, q_d + q_n) + \frac{q_e(q_n - 1)}{2^{32}} + \frac{q_d}{2^\tau},$$

where q_e is the number of encryption queries, q_d is the number of forgery attempts, such that $q_e + q_d \leq 2^{40}$, $q_n + q_d \leq 2^8$ is the maximum number of encryptions under the same (A, N) value, and t is the time complexity expressed in executions of ChiLow-$(32+\tau)$. Concretely, this bound implies the following limits. A single key can be used to encrypt up to 4 TB of data *in total*, which should suffice for all firmware updates during the lifetime of an embedded system. If the software encryption is done properly, that is, if the version counter

N is incremented every time a new binary object is encrypted, we have $q_n = 1$ and the third term vanishes. For example, if the code encryption key is dropped after 4 failed decryptions, each version counter could be used up to 64 times and the scheme would hold up, albeit with a degraded security level. The second term vanishes too if there are no forgery attempts and stays small compared to $\frac{q_d}{2^\tau}$ otherwise. Hence, security is not guaranteed only if the adversary can do an exhaustive key search or if the number of forgery attempts comes close to 2^τ.

We proceed similarly for ChiLow-40, although here the situation is simpler as we have a single tweakable block cipher.

Claim 2 *If keyed with a uniformly distributed secret key of* 128 *bits, and if* $q_t \leq 2^8$ *and* $q \leq 2^{40}$*, we claim that* D_{40} *satisfies*

$$\mathbf{Adv}_{D_{40}}^{\pm\widetilde{\mathrm{prp}}}(t, q, q_t) \leq \frac{t}{2^{128}},$$

where the adversarial resources are as in Claim 1.

Under Claim 2, Theorem 1 and Theorem 3, ChiLow-40 is an instantiation of ACE3 achieving **cae-r** and **cae-a** security up to the following advantages

$$\mathbf{Adv}_{\text{ChiLow-40}}^{\textbf{cae-r}}(q_e, q_d, q_n, t) \leq \frac{t}{2^{128}},$$
$$\mathbf{Adv}_{\text{ChiLow-40}}^{\textbf{cae-a}}(q_e, q_d, q_n, t) \leq \frac{t}{2^{128}} + \frac{q_e(q_n - 1)}{2^{40}} + \frac{q_d}{2^7}.$$

In the last expression, the first term represents the exhaustive key search, while the second and third terms are generic terms coming from Theorem 1. The limits are similar to those above, with the added value that ChiLow-40 supports the more robust security notion.

4 Design Rationale

4.1 Low-Latency Encryption Design Space

As pointed out in [31,12], the ultimate goal of a low-latency cryptographic primitive is to encrypt a block of data in a single clock cycle of a hardware implementation with a frequency as high as possible.

The first and most common approach for this was and, in some cases, still is an SPN structure with a cryptographically strong S-box of small dimension featuring a low latency hardware implementation; then combining it with a linear layer which locally is (almost) MDS. This approach is the foundation of some strong designs, such as Prince [12] and Qarma [4]. It usually also offers a low-latency decryption which can be taken as an advantage in several applications. However, in some recent designs, it is shown that if we are not restricted with low-latency in both directions, we can use more conceptual approaches to build ciphers with even lower latency. In Speedy [32] and in [40], it is explained how to build low-latency S-boxes of larger dimensions from a gate-level perspective, while providing good-enough cryptographic properties. Furthermore, Koala [21] achieves a low-latency design by applying functions of very large dimensions that cryptographically are not so strong in the forward direction but are very strong in the backward direction. As explained in the following, we combine these recent approaches for designing ChiLow.

4.2 The Nested Tweak-Key Schedule

Our block cipher and PRF support input sizes of 32 or 40 bits, while the tweak is 64-bit long and the master key has 128 bits. Absorbing all 192 bits into a 32- or 40-bit state would require at least 5 rounds, and an attacker could potentially guess three 32- or 40-bit blocks, allowing them to peel off at least 3 rounds. At the same time, ensuring a low latency forces us to use as few rounds as possible, both for the state update function itself as well as for processing the tweak and key. As a result, our design must meet the following constraints:

1. The latency of all paths must be equally low: the round function in the data path, tweak and key schedules must all have an almost equal and low latency;
2. Hard to exploit key guesses: the knowledge of some subkeys should not allow an attacker to easily recover information about the master key;
3. Fast diffusion: each bit of the state must depend (non-linearly) on the full input (plaintext, tweak and key) as quickly as possible; and
4. Even faster backward diffusion: since our low-latency constraint only applies in one direction, we focus on components with an extremely complex and high diffusion inverse.

To address the above requirements, we propose a nested construction inspired by the concept of encrypting the tweak with the master key. The state is divided into three components: the cipher state, the tweak state, and the key state. Each is updated using similar high-diffusion, low-latency non-linear round functions. The cipher state receives "subkeys" from the tweak state, while the tweak state gets its "subkeys" from the key state. As a consequence of this approach, an attacker trying to attack the cipher would need to guess either the material coming out of the tweak state or from the key state as discussed below.

The subkeys coming out from the tweak state into the cipher state are easy to exploit as they allow to directly peel off (partial) rounds of the encryption, but the information they provide has a complex relationship with the master key due to the fast diffusion in both the tweak state and the key state. Furthermore, each of these guesses is only relevant to a specific tweak, for which an attacker can only make a small number of queries.

The subkeys coming out of the key state are the same for all tweaks, and are more directly related to the master key. On the other hand, they enter the tweak state which is itself highly dependent on the master key. As a consequence, when looking at the last rounds, they will not facilitate the recovery of information about the tweak state bits.

The use of components with complex, high-degree, high-diffusion inverses means that the propagation of information or differential trails backwards can only yield useful data for one round.

4.3 The Non-Linear Layer

Our goal is to build a non-linear bijective function with minimal latency complexity where the latency complexity of a function is the minimum possible gate depth complexity of its implementation based on 2-bit `NAND`, 2-bit `NOR` and `INV` gates without counting the `INV` gates in the circuit paths [39, Section 3]. At the same time, we aim for non-trivial cryptographic properties. In [39], it is shown that there are very few *non-linear* and *balanced* functions with minimal latency complexity. Indeed, up to the extended bit-permutation equivalence, there is only one non-linear balanced function with latency complexity 2, namely $f(x_0, x_1, x_2) = x_0x_2 + x_1\overline{x_2}$ which can be seen as a multiplexer with x_2 being the selector for choosing one of x_0 and x_1 as the output. However, using only this Boolean function (and any other of its equivalent functions) as the coordinates of the bijective vectorial Boolean function, it is not possible to build any bijection with non-trivial linearity; i.e., for such a bijective mapping, there are always linear approximations with correlation ± 1 between the input and the output bits of the bijective vectorial Boolean function. Therefore, to build a non-linear layer with our criteria, a latency complexity of *at least 3 is necessary.*

As reported in [39], up to the extended bit-permutation equivalence, there are 2, 6, 8 and 3 Boolean functions with (full dependence on) 3, 4, 5 and 6 variables which have latency complexity 3. For further improvements of the latency, it is necessary to keep the fan-out of the gate in the previous layer low. In other words, the circuit of the non-linear layer should use each input variable as little as possible. With respect to this criterion, from the aforementioned 19 balanced Boolean functions with latency complexity 3, the one with representation $f(x_0, x_1, x_2) = x_0x_1 + x_2$ is the most suitable one: for each coordinate, only 3 inputs variables are involved, and in the circuit for each coordinate each variable is used only once. This suggests that it is possible to build a non-linear layer where each input bit is used only three times in the entire circuit of the non-linear layer. Indeed, the well-known χ functions, introduced by Daemen [16] and defined as follows, is a widely used example for this. For instance, χ_5 is used in KECCAK, χ_{257} in SUBTERRANEAN and KOALA, and χ_3 in several block ciphers such as 3-WAY.

Definition 1. *For n being an integer, $\chi_n \colon \mathbb{F}_2^n \to \mathbb{F}_2^n$ is defined by $y_i = x_i + \overline{x}_{i+1}x_{i+2}$ where the indices are taken modulo n.*

However, χ_n is bijective if and only if n is odd. Therefore, we cannot use it directly to design our primitives as we need non-linear layers of size 32, 40, 64, and 128. To solve this problem, we could use two concatenated χ functions. For m even, χ_{m-1} in parallel with χ_{m+1} gives a non-linear layer of input size $n = 2m$. Thereby, the inverse of the non-linear layer can be separated into two sub-functions of dimension $m - 1$ and $m + 1$ and each coordinate in the inverse direction will be dependent on $m-1$ or $m+1$ variables with an algebraic degree of either $\frac{m}{2}$ or $\frac{m}{2}+1$.

For our non-linear layers, we set out to improve over this. More precisely, we aim for a non-linear layer with the same nice cryptographic and implementation properties in the forward direction but with better (non-linear) diffusion and a higher algebraic degree in the inverse direction. Thereby, we restrict ourselves to coordinate functions of the form $y_i = x_{R(i)} + (x_{P(i)} + a_i)(x_{Q(i)} + b_i)$ where P, Q and R are permutations of $\mathbb{Z}_n$ and each a_i and b_i are Boolean constants. This ensures that each input variable will be used exactly three times.

To make the search for such a mapping with the aforementioned criteria as fast as possible, we omit redundant parts of the search. For instance, any permutation on the index of variables or any constant addition in the input or output of the function will end up with the same results. Hence, we fix that each $a_i = 1$. Furthermore, we ensure that the indices of the variables in the quadratic terms form ordered cycles. That is, for one cycle we have the quadratic terms $x_0x_1, x_1x_2, \ldots, x_{n-2}x_{n-1}, x_{n-1}x_0$. For two cycles, we have the quadratic terms $x_0x_1, x_1x_2, \ldots, x_{m-3}x_{m-2}, x_{m-2}x_0; x_{m-1}x_m, x_mx_{m+1}, \ldots, x_{n-2}x_{n-1}, x_{n-1}x_{m-1}$ for some integer m with $3 \leq m < n/2$.

In the case of a single cycle for the quadratic terms, for any choice of the permutation R and constants b_i, it is not possible to build a bijective function. The reason for this is that there will be a component function that is *bent*, and thus in particular not balanced. As being bent is invariant under adding linear functions, this cannot be compensated by xoring other inputs.

Therefore, we take the non-linear layer whose quadratic terms corresponds to two cycles, together with a constraint to split as equally as possible. Thereby, the function for the non-linear layer will be extended affine equivalent to $\chi_{m-1} \| \chi_{m+1}$. Consequently, this means that the linearity and differential uniformity of our non-linear layer stays the same as that of $\chi_{m-1} \| \chi_{m+1}$.

For $n \in \{8, 12, 16\}$, we exhaustively search through all permutation R and constants b_i to find non-linear bijections. Out of the many options (even after reducing up to the extended bit permutation equivalence), we chose the one already given in Eq. (1) as we could extend the pattern for these three small dimensions and show that the bijectivity is extendable to higher dimensions. We give another definition in Definition 2 that is based on χ. Metaphorically, we see our new non-linear layer as an intertwined version of two parallel χ functions, hence the name ChiChi or $\chi\!\chi$.

Definition 2. *For m even and $n = 2m$, we define $\chi\!\chi_n \colon \mathbb{F}_2^n \to \mathbb{F}_2^n$ as*

$$\chi\!\chi_n(x) = \begin{pmatrix} \chi_{m-1}(x_0, x_1, \ldots, x_{m-2}) \\ \chi_{m+1}(x_{m-1}, x_m, \ldots, x_{2m-1}) \end{pmatrix} + \lambda(x)$$

where the i-th coordinate of λ *is given by*

$$\lambda_i(x) = \begin{cases} x_m + x_{m-3} & \textit{if} \quad i = m-3 \\ x_{m-1} + x_{m-2} & \textit{if} \quad i = m-2 \\ x_{m-3} + x_m + x_{m-1} & \textit{if} \quad i = m-1 \\ x_m + x_{m-2} & \textit{if} \quad i = m \\ 0 & \textit{otherwise.} \end{cases}$$

Theorem 4. *For m even, $\chi\!\!\chi_{2m}$ is a bijection.*

Proof. Given $y = \chi\!\!\chi_{2m}(x)$, we give explicit formulas for x_m, $x_m + x_{m-3}$, x_{m-1}, and x_{m-2} as functions of the y_i variables in Lemmas 3 to 6 in Appendix D. Hence, given y, we can compute $\lambda(x)$ and consequently, invert the $\chi\!\!\chi_{2m}$ function since both χ_{m-1} and χ_{m+1} are invertible. □

Given the formula for x_m as a function of the variables y_i, we can present a lower bound on the algebraic degree of the inverse of $\chi\!\!\chi_{2m}$.

Corollary 1. *The inverse of $\chi\!\!\chi_{2m}$ has an algebraic degree of at least m.*

Proof. We give an explicit formula for x_m in Lemma 3. The algebraic normal form of x_m must contain exactly one monomial of degree m, namely $y_{m-4}\left(\prod_{k=1}^{\frac{m}{2}-2} y_{2k-1}\right) y_{m-2}\left(\prod_{k=1}^{\frac{m}{2}} y_{2k+m-1}\right)$. □

Regarding the algebraic degree of the other components, we can give essentially the same argument for the ones that we give explicitly in Appendix D. Below, we conjecture that the same actually holds for *every* non-trivial component of the inverse of $\chi\!\!\chi_{2m}$. This is based on our verification for dimensions up to $n = 32$ for n a multiple of 4.

Conjecture 1. Let m be an even integer. Every non-trivial component of the inverse of $\chi\!\!\chi_{2m}$ has algebraic degree m.

4.4 The Linear Layer

The linear layer plays a crucial role in achieving diffusion by ensuring that each input bit influences many output bits across the ciphertext. To meet this objective, we carefully select linear layers with strong diffusion properties. Additionally, we prioritize having a low latency: by ensuring that each output bit is the sum of three input bits, we can compute the linear layer and the round key addition with an `XOR` depth of two.

More precisely, all our linear layers $L_n : \mathbb{F}_2^n \mapsto \mathbb{F}_2^n$, with x mapping to y, are such that for every index i with $0 \leq i < n$:

$$y_i = x_{\alpha_0 i + \beta_0} + x_{\alpha_1 i + \beta_1} + x_{\alpha_2 i + \beta_2},$$

where indices are taken modulo n. In other words, each output bit depends on three input bits, where the indices of input bits are selected with an affine function A_L in $\mathbb{Z}_n$ that maps i to the tuple $(\alpha_0 i + \beta_0, \alpha_1 i + \beta_1, \alpha_2 i + \beta_2)$, with $1 \leq \alpha_j, \beta_j \leq n$ and each α_j being an odd integer. However, the search space for such linear layers is huge. To reduce the search space, we impose the following:

– We fix $\alpha_0 = \alpha_1 = \alpha_2 = \alpha$. Indeed, in our experiments for $n = 32$, we found that searches without such a restriction do not yield better results.
– We fix $\beta_0 < \beta_1 < \beta_2$, since the order of the tuples does not matter.

With these restrictions in place, the selection of A_L is done in two steps. First, we search for α and β_i values that ensure good diffusion properties. Specifically, after a certain number of iterations of $R_n = L_n \circ \chi\!\!\chi_n$, each output bit should depend on all n input bits – a property known as *full diffusion*. This property was checked using a symbolic evaluation of the rounds, and then checking whether each output bit is a function of all the input bits.

The number of iterations required for full diffusion varies with the value of n. We then select L_n candidates that have the fastest diffusion, i.e., the number of rounds needed for full diffusion is the lowest. As it turns out, we need only two rounds for $n = 32$, three rounds for $n \in \{40, 64\}$, and four rounds for $n = 128$.

Then, in the second step, we go through the remaining candidates from the previous step and select the ones that performs the best across 3 and 4 rounds in terms of differential probabilities and linear correlations.

4.5 The Round Constants

In order to prevent slide attacks, we simply add some round constants in the master key path. To make sure that the Hamming weights of their differences are not too low, we combine a counter with a single bit that is shifted for each round. An additional bit which is set for ChiLow-40 provides domain separation. In the end, the round constant used in round i is

$$c^{(i)}_{96\ldots127} = i \oplus (1 \ll (i+4)) \oplus (b \ll 31) , \tag{2}$$

where $\ll$ denotes a left shift, and $b \in \{0, 1\}$ is set to 1 only for ChiLow-40.

5 Security Analysis

In this section, we provide details on the security analysis of ChiLow. For this, we take advantage of the existing analyses on the designs with similar structures.

From the provided analyses below, we deduce that 8-round ChiLow is secure under the given security model (which in particular implies restrictions on the data complexity).

5.1 Differential and Linear Distinguishers

Since $\chi\!\!\chi$ is extended affine equivalent to two parallel χ functions, we can use all the knowledge on the differential and linear transitions over this mapping. For

Table 2: The max DP (in $-\log_2(\cdot)$) of a single trail for case (X) that the data path of each primitive and the tweak schedule alone (left), and for case $(\mathsf{X}+\mathsf{T})$ that considers the related-tweak trails with input difference only in the tweak state (center), and the maximum square correlation (in $-\log_2(\cdot)$) of a single trail for case (X) that the data path of each primitive alone (right).

Round	(X)				$(\mathsf{X}+\mathsf{T})$			(X)	
	D_{32}	D'_{32}	D_{40}	T	D_{32}	D'_{32}	D_{40}	D_{32}	D_{40}
1	2	2	2	2	2	2	2	2	2
2	8	8	7	8	8	8	8	8	8
3	19	18	19	22	33	33	34	14	16

Table 3: The max DP for the $(\mathsf{X}+\mathsf{T})$ case (in $-\log_2(\cdot)$) of differentials considering clustering. For each number of round and each case, we report on the maximum DP of single trail following the differential (P), the clustering effect (CE) observed and differential probability (DP).

Round	D_{32}			D'_{32}			D_{40}		
	P	CE	DP	P	CE	DP	P	CE	DP
1	2	–	2	2	–	2	2	–	2
2	8	2	6	8	2	6	8	2	6
3	33	8	26	33	8	26	34	4	30

this purpose, we use and modify the properties from [34] and [35] to evaluate the resistance of ChiLow against differential and linear cryptanalysis.

We use the Mixed Integer Linear Programming (MILP) code from [21] that was used for the differential and linear security analysis of Koala, and modify it for a single χ function. We then use it to model the propagation of differential and linear trails over several rounds of ChiLow. More detail can be found in E.

To take the security model specific to our use cases into account, we model and report on different type of trails, including differential trails for a single path (X) (without considering the differences in the tweak/key addition) in Table 2 (left), and related-tweak differential trails where there is an input difference only in the tweak state $(\mathsf{X}+\mathsf{T})$ in Table 2 (right). We also report on the exact trails for $(\mathsf{X}+\mathsf{T})$ in E.3. Note that due to the restriction on the number of queries that an adversary can request for the same tweak and Section 2, we do not consider the trails with non-zero differences in both the tweak state and the cipher state.

Clustering Effect We analyze the clustering effect on the differential probability for our constructions ChiLow-(32+τ) and ChiLow-40. To identify clustering effects, we reuse the model designed to find the single trail with the maximum differential probability and leverage a feature, namely the PoolSearchMode-based

Table 4: The bit differences are shown after each operation of a 3-round differential trail of ChiLow-$(32+\tau)$.

	Data Path	Tweak Path
Δ_{in}	....	1
χ	....	1
L_{32}	....	....1....1.......1........
$\oplus$	....	
χ	....	....1....1.....1.1........
L_{32}	....	.1.....1.1........1....1.......1...1....1.......1.........1.1..1
$\oplus$	.1.....1.1........1....1.......1	
χ	.1...1.1.1....1...1...11.....1.1	.1...111.1......111..1.1......1..111..1.1......11.......111.1.1.
L_{32}	1.11.......111.11.11...11.11.1.1	1..1.......11..11..1...11.11.1.111...1....1...1.11.......1111..1
$\oplus$	..1..........1....1.............	

solution finding approach[10] of the Gurobi solver that allows us to search not only for the optimal solution but also for a set of the top t solutions, where t is a parameter we can define. Once we have this set of t best single trails, we fix the input and output differences and rerun the solver, incorporating these as constraints to search for the top l solutions under the new conditions. For each input-output difference pair, this process yields a distribution of trails according to the maximum differential probabilities. From this distribution, we can compute the clustering effect for the given differential. Wherever possible, we also validate our results through experimental verification.

We begin by examining the clustering effect on ChiLow-$(32+\tau)$, where a strong clustering effect is evident as shown in Table 3. For instance, the best differential probability observed for 3 rounds of ChiLow-$(32+\tau)$ is 2^{-33}. However, when clustering is considered, the probability increases to 2^{-26}. The main reason behind this is that, during each round, the lower 32 bits of the 64-bit tweak state are used in D_{32}, and the upper 32 bits in D'_{32}. Therefore, the differences in the upper 32 bits of the tweak do not affect the data path. For example, if no active bits are present in the lower 32-bit of the tweak state during the first two rounds, the difference propagation in the first two rounds of D_{32} occurs with probability 1 – a free differential propagation. Such a trail is shown in Table 4. Furthermore, multiple trails can arise from a given input tweak difference, that only affect the upper part of the tweak state, which is the primary source of the clustering effect. However, despite this clustering effect, considering the data limit due to the construction, the maximum differential probability of any trail for ChiLow-$(32+\tau)$ is sufficiently low when we consider more than 4 rounds.

We also analyze the clustering effect on ChiLow-40. Its clustering effect is lower than that in ChiLow-$(32+\tau)$ as it takes more bits from the tweak state.

We also analyze the security of ChiLow-$(32+\tau)$ and ChiLow-40 against linear cryptanalysis. To model the linear propagation trough χ we choose a lazy approach, leading to possible false positive results. This allows us to use a few equations and obtain a rather simple and efficient model. As for the differential

[10] https://www.gurobi.com/documentation/current/refman/poolsearchmode.html

trail, we report in Table 2, the maximum square correlation found for a single trail, for the data path of each primitive.

Multiple-Tweak Differential Attack In their analysis of SCARF [13], Boura *et al.* carefully studied the differentials with probability close to the natural bound. More precisely, they accurately computed the bias ϵ of differentials of probability $1/(2^n-1)+\epsilon$ where n is the block size. Distinguishing a random permutation for which $\epsilon=0$ requires around $\epsilon^{-2}2^n$ pairs when ϵ is much lower than p. Unfortunately it was not possible to compute the bias as they did in SCARF since the block sizes of our primitives is too large. Instead we experimentally searched for differentials for the function D_{32} used in ChiLow-$(32+\tau)$ with only one active bit at the input. We encrypted 2^{40} pairs for each position of the active bit and looked at the best differentials for 5 rounds. We were unable to observe anything that would permit to distinguish D_{32} from a random permutation. Thus, according to the restriction we set on the amount of data that can be encrypted under the same master key, we claim that our construction is secure against this type of attack.

5.2 Attacks Based on the ANF

While very efficient, χ and $\chi\!\!\chi$ functions are only of degree 2 and thus it is important to carefully analyze the resistance of our construction against algebraic attacks. The primitives relying on the χ function as the source of non-linearity most often involve a large number of rounds to compensate for its low algebraic degree while we only use 8 rounds.

The main question is to determine the algebraic degree of the $D_{32}(K,T,X)$ function. We experimentally verified by computing the ANF representation of the output for random values of (K,T) that every monomial of degree 31 in variables from X does appear in the representation. More precisely, given u of hamming weight 31, we can write the i-th coordinate of $D_{32}(K,T,X)$ as $p_{u,i}(K,T,X)X^u \oplus q_{u,i}(K,T,X)$ where $p_{u,i}$ and $q_{u,i}$ both are polynomials and such that X^u does not divide any monomial of $q_{u,i}$. We prove that the $32\times 32 = 256$ polynomials $p_{u,i}$ are non-zero and linearly independent after 6 rounds. This ensures that there is no integral distinguisher on more than 5 rounds of the cipher in the single-key single-tweak model.

To study the algebraic resistance of our primitive in the related-tweak setting, we used division property without unknown subsets, to track the biggest monomials involving both tweak and key variables. However, there are too many division trails to computationally prove that there is no integral distinguisher on more than 6 rounds in the single-key multiple-tweaks model. Still, regarding the limitations we have on the number of messages and tweaks that can be encrypted under the same key, we claim that 8 rounds are fully secure against any algebraic and integral attack in the single key model.

5.3 Meet-in-the-Middle Attacks

The meet-in-the-middle (MITM) attack has proven to be a powerful technique, leading to numerous successful attacks on both block ciphers and hash functions [18,8,19]. Recent works have applied the MITM technique to preimage attacks on Keccak and Ascon [38,20], both of which utilize the non-linear χ operation in their designs. As ChiLow employs variants of the χ operation, we try to mount MITM key-recovery attacks on it.

The fundamental principle of the MITM key-recovery attack involves partitioning the key state into two independent parts (under certain constrains). By conducting exhaustive searches for the key, we can match the results from these two parts. As shown in Fig. 4, the attack can be visualized by a coloring scheme for the bits in the state: computations involving blue bits are independent of those involving red bits. The gray bits remain constant, while the white bits are unknown in the computation based on either the red or blue bits. Importantly, the yellow bits represent linear combinations of the bits in red and blue. Although their values are unknown on each side, they can still facilitate matching (the blue and red bits satisfy some linear equations) during the attack. Before applying the round functions to the states, we impose some constraints on certain bits of the states such that the specified coloring scheme is fulfilled. For example, according to Fig. 4, the 30th bit in first row of K_1 turns from red to gray (see $K^\dagger$), which means there is a (non-linear) constraint on the red key bits such that the 30th bit in the first row is fixed to a constant. The number of constraints should be less than the number of bits with same color in the initial state, otherwise there might be a overdetermined system on key bits that leads to a contradiction. To enhance our approach, we utilize an automated search method and have developed an SAT-based search tool specifically for MITM attacks on ChiLow. Through our investigations, for ChiLow-$(32 + \tau)$, we have identified a 4-round key-recovery attack that operates with 2-bit advantage (Fig. 6) and a 3-round attack with 22-bit advantage (Fig. 4, Alg. 3). In the 3-round attack (Alg. 3), we compute ChiLow-$(32 + \tau)$ about 2^{103} times in line 5-18 and 2^{102} times in line 16-25, the total time complexity is $2^{2+104} = 2^{106}$, and the memory complexity is 2^{103}. Similarly, the total time complexity of 4-round attack is 2^{126}. For ChiLow-40, we have identified a 4-round key-recovery attack that operates with 2-bit advantage (Fig. 8) and a 3-round attack with 22-bit advantage (Fig. 7). The time complexities of the two attacks are 2^{126} and 2^{106} respectively. We note that these attacks might not be optimal since our tool did not exhausted the search space. We provide more details for mounting a MITM key-recovery attack on ChiLow-$(32 + \tau)$ in Appendix F.

6 Hardware Evaluation

We focus on the performance evaluation of the decryption functionality, as encryption can be performed offline and thus its performance is less critical.

To achieve low latency, we implemented our schemes in a fully unrolled circuit, i.e., by replicating and chaining the round function logic. This allows us

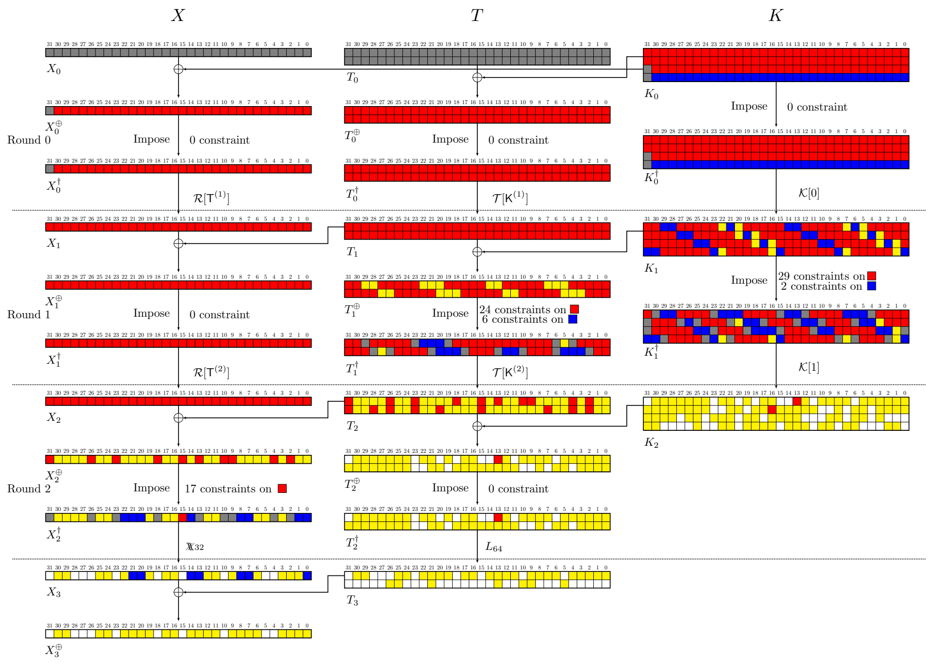


Fig. 4: MITM key-recovery attack on 3-round ChiLow-$(32 + \tau)$.

to evaluate the whole decryption in a single clock cycle. We implemented two variants for each scheme, one that includes the key schedule and one that does not. In the latter case, the round keys are precomputed and given as input to the circuit. Note that by construction, only 64 bits of each round key are used to update the tweak, therefore the number of key bits given as input is $(9 \times 64 =)$ 576 for ChiLow-$(32 + \tau)$ and $(8 \times 64 + 40 =)$ 552 for ChiLow-40.

We compare the performance of ChiLow with other low-latency designs. To the best of our knowledge, there are no low-latency schemes that target code encryption specifically or that have a block size suitable for our use case. First, we compare with the tweakable block cipher Scarf, whose block size is only 10 bits, resulting in very small area and power consumption. Then, we consider the block cipher Prince as well as the tweakable block cipher Qarmav1-64, both working on a 64-bit cipher state. For Qarmav1, we consider $r = 5$ and σ_0, which is the configuration recommended for Pointer Authentication Code (PAC) or Memory Integrity Applications (MIA). We do not include Princev2 and Qarmav2 in the comparison, as their latency and area are almost the same as the original designs. Additionally, we consider the low-latency PRF Orthros. We do not include the PRF Gleeok since Orthros outperforms it in both area and latency. However, the authors of Gleeok also propose an AE scheme based on it that targets memory encryption. It composes the counter (CTR) mode with two independent instances of the Inner Product (IP) hashing, which makes use of multiplications over $\mathbb{F}_{2^s}$. We consider the smallest variant, called Gleeok-128-IP$_{64}$, where two multipliers over $\mathbb{F}_{2^{64}}$ are used. Note that during decryption, the PRF cores and the multipliers work independently, with the critical path given by the PRF component. For Prince, Orthros, and Gleeok-128-IP$_{64}$, we made use of the HDL code publicly available at [7,2].

Methodology. All ciphers are implemented as fully combinational circuits.The RTL codes are synthesized using Synopsys Design Compiler version V-2023.12 and the Nangate 15nm open cell library. We ask the compiler to constrain the delay between input/output ports to progressively lower values until the tool cannot produce a circuit satisfying the constraint. We ran post-synthesis simulation at a frequency of 10MHz, and collected the switching activity of each gate of the circuit. Finally, we obtained the average power consumption using Synopsys Prime Time version V-2023-12, using the back annotated switching activity and we derived energy consumption from it.

Results. Experimental results are given in Table 5. We include performance figures only for ChiLow-$(32 + 16)$, as the difference with ChiLow-$(32 + 8)$ is negligible. Area figures do not take into account the cost of storing the round keys in registers, as all ciphers are implemented as fully combinational circuits. For the Nangate 15nm library, the cost of storing 128, 240, and 256 bits of key is $163.58\mu\text{m}^2$, $306.71\mu\text{m}^2$, and $327.16\mu\text{m}^2$, respectively. When ChiLow-$(32+\tau)$ and ChiLow-40 are implemented without the key schedule circuit, the cost of storing the precomputed key material is $705.43\mu\text{m}^2$ and $736.10\mu\text{m}^2$, respectively. Precomputing the round keys for ChiLow reduces the total area cost without impacting the latency, making this approach ideal when area is a critical aspect.

To demonstrate the full potential of the selected ciphers, power and energy costs are evaluated using a fixed key, as in our use case the decryption key will rarely change, reducing the switching activity of the key-schedule circuits. The experimental results show how ChiLow outperforms the other ciphers in terms of latency, power and energy cost. The only exception is Scarf, which however has a block size of only 10 bits. In terms of area, Prince, Qarma and Scarf achieve in general better results. However, ChiLow provides also authentication which these ciphers do not. Gleeok-128-IP$_{64}$ offers authentication, but the use of large finite field multipliers results in a high area cost and as a consequence in high power and energy.

The hardware evaluation confirms that, compared to other solutions, ChiLow is more suitable for on-the-fly code decryption and authentication, as it achieves at the same time low latency, low power and low energy.

7 Conclusion and Future Work

In this work, we modeled the real-world problem of code encryption and presented a practical solution with strong performance and innovative design ideas. We feel that our solution is a significant and sound step forward, that also triggers several directions for future research. Concerning our new non-linear layer

Table 5: Comparison results for the Nangate 15nm OCL at a clock frequency of 10 MHz. The key is fixed according to the application. * refers to the implementation without the key schedule.

Cipher	Type	Key/Tweak/Block	Area		Latency	Power	Energy
		[bits/bits/bits]	[μm^2]	[GE]	[ps]	[mW]	[pJ]
ChiLow-$(32+16)$	TBC+PRF	128/64/32	3417.93	17384.49	302.03	0.2444	24.44
			3581.85	18218.25	282.73	0.2521	25.21
ChiLow-40	TBC	128/64/40	3043.49	15480.00	299.21	0.2103	21.03
			3205.20	16302,50	278.31	0.1997	19.97
ChiLow-$(32+16)$ *	TBC+PRF	128/64/32	2004.17	10193.75	301.78	0.2252	22.52
			2104.20	10702.50	285.84	0.2178	21.78
ChiLow-40*	TBC	128/64/40	1639.86	8340.75	298.80	0.1832	18.32
			1734.33	8821.25	280.60	0.1710	17.10
Scarf	TBC	240/48/10	968,88	4928,00	297,89	0.0896	8.96
			986,73	5018,75	248,91	0.0749	7.49
			1096,48	5577,00	216,80	0.0793	7.93
Prince	BC	128/-/64	1798.96	9150.00	450.00	0.2955	29.55
			2090.39	10632.25	400.00	0.3267	32.67
			2450.08	12461.75	374.04	0.3433	34.33
Qarmav1-64	TBC	128/64/64	1912.21	9726.00	537.39	0.7025	70.25
$(r=5)$			1963.03	9984.50	450.00	0.6010	60.10
			2652.09	13489.25	378.30	0.8172	81.72
Orthros	PRF	128/-/128	5850.42	29756.75	448.99	0.9791	97.90
			6123.31	31144.75	398.99	0.7400	74.00
			7372.26	37497.25	354.79	0.8144	81.40
Gleeok-128-IP$_{64}$	AE	256/-/128	23489.35	119473.01	450.00	4.3000	430.00
			23703.80	120563.76	400.00	4.0440	404.40
			24616.70	125207.01	382.99	4.1930	419.30

ChiChi, we see three clear paths to continue: proving Conjecture 1 to assess its resistance against integral attacks; constructing a similar family for even dimensions not divisible by four; exploring its potential for other applications. Concerning ChiLow, but also ChiChi, a challenge is adding side-channel protections. While most of the approaches for masking ciphers based on χ are valid also for our primitives, doing so within *one* clock-cycle remains an open problem, not only for our designs but in general. While the `TWEAKEY` framework [29] already provides a nice solution, we expect that studying our nested tweak-key schedule in a more general setup and deriving its properties might turn it into an alternative for combining tweak and key. Finally, as for any new design, we encourage more cryptanalysis on the primitives, especially making use of the small block size to develop a potentially improved version of the recently published attack on Scarf [13].

Acknowledgments

This work started during the Lorentz Center workshop on "Beating Real-Time Crypto: Solutions and Analysis", held in Leiden, the Netherlands, in April 2024, and we thank the organizers for inviting us. We thank Ruggero Susella for his constructive comments while preparing this work. This work was (in part) supported by: the Crypto Frontiers Research Center of Intel Labs, the European Research Council (ERC) project 101097056 (SYMTRUST), the Deutsche Forschungsgemeinschaft (DFG) project for Germany's Excellence Strategy (EXC 2092 CASA) 390781972, the Dutch Research Council (NWO) project PROACT (NWA.1215.18.014) and the CiCS project of the research programme Gravitation (024.006.037), the European Research Council (ERC) project 101041545 (ReSCALE), the Agence Nationale de la Recherche through the SWAP project under Contract ANR-21-CE39-0012 and through the OREO project under Contract ANR-22-CE39-0015, the France 2030 program under grant agreement No. ANR-22-PECY-0010, the European Research Council (ERC) project 101125450 (SoBaSyC), the National Key Research and Development Program of China (2022YFB2701900), the National Natural Science Foundation of China (62032014), the National Cryptologic Science Foundation of China.

References

1. Anand, R., Banik, S., Caforio, A., Ishikawa, T., Isobe, T., Liu, F., Minematsu, K., Rahman, M., Sakamoto, K.: Gleeok: A family of low-latency PRFs and its applications to authenticated encryption. IACR TCHES **2024**(2), 545–587 (2024). https://doi.org/10.46586/tches.v2024.i2.545-587
2. Anand, R., Banik, S., Caforio, A., Ishikawa, T., Isobe, T., Liu, F., Minematsu, K., Rahman, M., Sakamoto, K.: Gleeok: A family of low-latency prfs and its applications to authenticated encryption (implementations). https://github.com/qantik/gleeok (2024)
3. Andreeva, E., Bhati, A.S., Preneel, B., Vizár, D.: 1, 2, 3, fork: Counter mode variants based on a generalized forkcipher. IACR Trans. Symm. Cryptol. **2021**(3), 1–35 (2021). https://doi.org/10.46586/tosc.v2021.i3.1-35
4. Avanzi, R.: The QARMA block cipher family. IACR Trans. Symm. Cryptol. **2017**(1), 4–44 (2017). https://doi.org/10.13154/tosc.v2017.i1.4-44
5. Avanzi, R., Banik, S., Dunkelman, O., Eichlseder, M., Ghosh, S., Nageler, M., Regazzoni, F.: The QARMAv2 family of tweakable block ciphers. IACR Trans. Symm. Cryptol. **2023**(3), 25–73 (2023). https://doi.org/10.46586/tosc.v2023.i3.25-73
6. Banik, S., Isobe, T., Liu, F., Minematsu, K., Sakamoto, K.: Orthros: A low-latency PRF. IACR Trans. Symm. Cryptol. **2021**(1), 37–77 (2021). https://doi.org/10.46586/tosc.v2021.i1.37-77
7. Banik, S., Isobe, T., Liu, F., Minematsu, K., Sakamoto, K.: Orthros: A low-latency PRF (implementations). https://github.com/subhadeep-banik/orthros (2021)
8. Bao, Z., Dong, X., Guo, J., Li, Z., Shi, D., Sun, S., Wang, X.: Automatic search of meet-in-the-middle preimage attacks on aes-like hashing. In: Canteaut, A., Standaert, F. (eds.) Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12696, pp. 771–804. Springer (2021). https://doi.org/10.1007/978-3-030-77870-5_27, https://doi.org/10.1007/978-3-030-77870-5_27

9. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part II. LNCS, vol. 9815, pp. 123–153. Springer, Berlin, Heidelberg (Aug 2016). `https://doi.org/10.1007/978-3-662-53008-5_5`
10. Belkheyar, Y., Daemen, J., Dobraunig, C., Ghosh, S., Rasoolzadeh, S.: BipBip: A low-latency tweakable block cipher with small dimensions. IACR TCHES **2023**(1), 326–368 (2023). `https://doi.org/10.46586/tches.v2023.i1.326-368`
11. Bellare, M., Rogaway, P.: Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 317–330. Springer, Berlin, Heidelberg (Dec 2000). `https://doi.org/10.1007/3-540-44448-3_24`
12. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knežević, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçin, T.: PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 208–225. Springer, Berlin, Heidelberg (Dec 2012). `https://doi.org/10.1007/978-3-642-34961-4_14`
13. Boura, C., Rasoolzadeh, S., Saha, D., Todo, Y.: Multiple-tweak differential attack against SCARF. Cryptology ePrint Archive, Paper 2024/1408 (2024), `https://eprint.iacr.org/2024/1408`
14. Bozilov, D., Eichlseder, M., Knezevic, M., Lambin, B., Leander, G., Moos, T., Nikov, V., Rasoolzadeh, S., Todo, Y., Wiemer, F.: PRINCEv2 - more security for (almost) no overhead. In: Dunkelman, O., Jr., M.J.J., O'Flynn, C. (eds.) SAC 2020. LNCS, vol. 12804, pp. 483–511. Springer, Cham (Oct 2020). `https://doi.org/10.1007/978-3-030-81652-0_19`
15. Chakraborti, A., Datta, N., Jha, A., Mancillas-López, C., Nandi, M., Sasaki, Y.: Elastic-tweak: A framework for short tweak tweakable block cipher. In: Adhikari, A., Küsters, R., Preneel, B. (eds.) INDOCRYPT 2021. LNCS, vol. 13143, pp. 114–137. Springer, Cham (Dec 2021). `https://doi.org/10.1007/978-3-030-92518-5_6`
16. Daemen, J.: Cipher and hash function design, strategies based on linear and differential cryptanalysis, PhD Thesis. K.U.Leuven (1995), `http://jda.noekeon.org/`
17. Daemen, J., Massolino, P.M.C., Mehrdad, A., Rotella, Y.: The subterranean 2.0 cipher suite. IACR Trans. Symmetric Cryptol. **2020**(S1), 262–294 (2020). `https://doi.org/10.13154/TOSC.V2020.IS1.262-294`, `https://doi.org/10.13154/tosc.v2020.iS1.262-294`
18. Diffie, W., Hellman, M.E.: Special feature exhaustive cryptanalysis of the NBS data encryption standard. Computer **10**(6), 74–84 (1977). `https://doi.org/10.1109/C-M.1977.217750`, `https://doi.org/10.1109/C-M.1977.217750`
19. Dong, X., Hua, J., Sun, S., Li, Z., Wang, X., Hu, L.: Meet-in-the-middle attacks revisited: Key-recovery, collision, and preimage attacks. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part III. Lecture Notes in Computer Science, vol. 12827, pp. 278–308. Springer (2021). `https://doi.org/10.1007/978-3-030-84252-9_10`, `https://doi.org/10.1007/978-3-030-84252-9_10`
20. Dong, X., Zhao, B., Qin, L., Hou, Q., Zhang, S., Wang, X.: Generic mitm attack frameworks on sponge constructions. In: Reyzin, L., Stebila, D. (eds.) Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference,

Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 14923, pp. 3–37. Springer (2024). https://doi.org/10.1007/978-3-031-68385-5_1, https://doi.org/10.1007/978-3-031-68385-5_1

21. Eliasi, P.A., Belkheyar, Y., Daemen, J., Ghosh, S., Kuijsters, D., Mehrdad, A., Mella, S., Rasoolzadeh, S., Van Assche, G.: Koala: A low-latency pseudorandom function. In: Selected Areas in Cryptography (2024)
22. Forler, C., List, E., Lucks, S., Wenzel, J.: Reforgeability of authenticated encryption schemes. In: Pieprzyk, J., Suriadi, S. (eds.) ACISP 17, Part II. LNCS, vol. 10343, pp. 19–37. Springer, Cham (Jul 2017). https://doi.org/10.1007/978-3-319-59870-3_2
23. Gilboa, S., Gueron, S., Morris, B.: How many queries are needed to distinguish a truncated random permutation from a random function? Journal of Cryptology **31**(1), 162–171 (Jan 2018). https://doi.org/10.1007/s00145-017-9253-0
24. Greene, P., Motley, M., Weeks, B.: ARADI and LLAMA: Low-latency cryptography for memory encryption. Cryptology ePrint Archive, Report 2024/1240 (2024), https://eprint.iacr.org/2024/1240
25. Hall, C., Wagner, D., Kelsey, J., Schneier, B.: Building PRFs from PRPs. In: Krawczyk, H. (ed.) CRYPTO'98. LNCS, vol. 1462, pp. 370–389. Springer, Berlin, Heidelberg (Aug 1998). https://doi.org/10.1007/BFb0055742
26. Hoang, V.T., Krovetz, T., Rogaway, P.: Robust authenticated-encryption AEZ and the problem that it solves. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 15–44. Springer, Berlin, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46800-5_2
27. Hoang, V.T., Reyhanitabar, R., Rogaway, P., Vizár, D.: Online authenticated-encryption and its nonce-reuse misuse-resistance. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 493–517. Springer, Berlin, Heidelberg (Aug 2015). https://doi.org/10.1007/978-3-662-47989-6_24
28. IEEE: Standard for cryptographic protection of data on block-oriented storage devices. IEEE Std 1619-2018 (Revision of IEEE Std 1619-2007) (2019)
29. Jean, J., Nikolic, I., Peyrin, T.: Tweaks and keys for block ciphers: The TWEAKEY framework. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 274–288. Springer, Berlin, Heidelberg (Dec 2014). https://doi.org/10.1007/978-3-662-45608-8_15
30. Jean, J., Nikolic, I., Peyrin, T., Seurin, Y.: The deoxys AEAD family. Journal of Cryptology **34**(3), 31 (Jul 2021). https://doi.org/10.1007/s00145-021-09397-w
31. Knežević, M., Nikov, V., Rombouts, P.: Low-latency encryption - is "lightweight = light + wait"? In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 426–446. Springer, Berlin, Heidelberg (Sep 2012). https://doi.org/10.1007/978-3-642-33027-8_25
32. Leander, G., Moos, T., Moradi, A., Rasoolzadeh, S.: The SPEEDY family of block ciphers engineering an ultra low-latency cipher from gate level for secure processor architectures. IACR TCHES **2021**(4), 510–545 (2021). https://doi.org/10.46586/tches.v2021.i4.510-545, https://tches.iacr.org/index.php/TCHES/article/view/9074
33. Leurent, G., Sibleyras, F.: The missing difference problem, and its applications to counter mode encryption. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 745–770. Springer, Cham (Apr / May 2018). https://doi.org/10.1007/978-3-319-78375-8_24

34. Mehrdad, A., Mella, S., Grassi, L., Daemen, J.: Differential trail search in cryptographic primitives with big-circle chi: Application to Subterranean. IACR Trans. Symm. Cryptol. **2022**(2), 253–288 (2022). `https://doi.org/10.46586/tosc.v2022.i2.253-288`
35. Mella, S., Mehrdad, A., Daemen, J.: Differential and linear properties of vectorial boolean functions based on chi. Cryptogr. Commun. **15**(6), 1087–1116 (2023). `https://doi.org/10.1007/S12095-023-00639-1`, `https://doi.org/10.1007/s12095-023-00639-1`
36. Namprempre, C., Rogaway, P., Shrimpton, T.: Reconsidering generic composition. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 257–274. Springer, Berlin, Heidelberg (May 2014). `https://doi.org/10.1007/978-3-642-55220-5_15`
37. National Institute of Standards & Technology: Recommendation for block cipher modes of operation: the XTS-AES mode for confidentiality on storage devices. NIST Special Publication 800-38E (2010)
38. Qin, L., Hua, J., Dong, X., Yan, H., Wang, X.: Meet-in-the-middle preimage attacks on sponge-based hashing. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 14007, pp. 158–188. Springer (2023). `https://doi.org/10.1007/978-3-031-30634-1_6`, `https://doi.org/10.1007/978-3-031-30634-1_6`
39. Rasoolzadeh, S.: Low-latency boolean functions and bijective S-boxes. IACR Trans. Symm. Cryptol. **2022**(3), 403–447 (2022). `https://doi.org/10.46586/tosc.v2022.i3.403-447`
40. Rasoolzadeh, S.: Classification of all t-resilient boolean functions with t + 4 variables. IACR Trans. Symm. Cryptol. **2023**(3), 213–226 (2023). `https://doi.org/10.46586/tosc.v2023.i3.213-226`
41. Rogaway, P.: Authenticated-encryption with associated-data. In: Atluri, V. (ed.) ACM CCS 2002. pp. 98–107. ACM Press (Nov 2002). `https://doi.org/10.1145/586110.586125`
42. STMicroelectronics: How to use OTFDEC for encryption/decryption in trusted environment on STM32H7Bxxx and STM32H73xx microcontrollers. STMicroelectronics Application Note AN5281 (2020)
43. Ueno, R., Haneda, H., Homma, N., Inoue, A., Minematsu, K.: Crystalor: Recoverable memory encryption mechanism with optimized metadata structure. Cryptology ePrint Archive, Report 2023/1630 (2023), `https://eprint.iacr.org/2023/1630`
44. Wang, J., Huang, T., Wu, S., Liu, Z.: Twinkle: A family of low-latency schemes for authenticated encryption and pointer authentication. IACR Commun. Cryptol. **1**(2), 20 (2024). `https://doi.org/10.62056/A3N59QGXQ`, `https://doi.org/10.62056/a3n59qgxq`
45. Werner, M., Unterluggauer, T., Schilling, R., Schaffenrath, D., Mangard, S.: Transparent memory encryption and authentication. In: Santambrogio, M.D., Göhringer, D., Stroobandt, D., Mentens, N., Nurmi, J. (eds.) 27th International Conference on Field Programmable Logic and Applications, FPL 2017, Ghent, Belgium, September 4-8, 2017. pp. 1–6. IEEE (2017). `https://doi.org/10.23919/FPL.2017.8056797`, `https://doi.org/10.23919/FPL.2017.8056797`

A More Details on the Constructions

ACE1. ACE1 is parameterized with an n-bit PRF f (used as a stream cipher) and a τ-bit PRF F. Being a pragmatic instance of the Encrypt-then-MAC (EtM) generic composition [36], it allows the plaintext decryption and tag verification to be parallelized. ACE1 may be instantiated with a single primitive (PRF or a TBC) using a single secret key, provided a domain separation bit is reserved in the tweak.

If $q_n = 1$, ACE1 is **cae-a** secure, see Theorem 5. Care must be taken with small n. For example, implementing the stream cipher $\mathsf{f}(K, A, N) = \mathsf{E}(K, N, A)$ with a short-tweak tweakable block cipher E [15], where presumably shorter input N is passed as a tweak and the larger address A is passed as the message input, the term $\mathbf{Adv}_{\mathsf{f}}^{\mathrm{prf}}(q_e, t) \approx \mathbf{Adv}_{\mathsf{E}}^{\widetilde{\mathrm{prp}}}(q_e, t) + q_e^2/2^{n+1}$ can lead to practical attacks with $q_e \approx 2^n$ [33] (256 kB when $n = 32$). Instead, a beyond-birthday secure tweakable block cipher counter mode variant ought to be used, e.g., with entire (A, N) in the tweak [3].

```
algorithm ACE1.$\mathcal{E}(K, N, A, M)$
Parse $K_1, K_2 \leftarrow K$
$\bar{C} \leftarrow M \oplus \mathsf{f}(K_1, A\|N)$
$T \leftarrow \mathsf{F}(K_2, A\|N, C)$
return $\bar{C}\|T$
```

```
algorithm ACE1.$\mathcal{D}(K, N, A, C)$
Parse $K_1, K_2 \leftarrow K$
Set $C \leftarrow C_{0\cdots n-1}$, $T \leftarrow C_{n\cdots n+\tau-1}$
if $\mathsf{F}(K_2, A\|N, C) \neq T$ then
  return $\perp$
$M \leftarrow C \oplus \mathsf{f}(K_1, A\|N)$
return $M$
```

```
algorithm ACE2.$\mathcal{E}(K, N, A, M)$
Parse $K_1, K_2 \leftarrow K$
$\bar{C} \leftarrow \mathsf{E}(K_1, A\|N, M)$
$T \leftarrow \mathsf{F}(K_2, A\|N, C)$
return $\bar{C}\|T$
```

```
algorithm ACE2.$\mathcal{D}(K, N, A, C)$
Parse $K_1, K_2 \leftarrow K$
Set $C \leftarrow C_{0\cdots n-1}$, $T \leftarrow C_{n\cdots n+\tau-1}$
if $\mathsf{F}(K_2, A\|N, C) \neq T$ then
  return $\perp$
$M \leftarrow \mathsf{E}^{-1}(K_1, A\|N, C)$
return $M$
```

```
algorithm ACE3.$\mathcal{E}(K, N, A, M)$
$\bar{M} \leftarrow M\|0^\tau$
$C \leftarrow \mathsf{E}(K, A\|N, M)$
return $C$
```

```
algorithm ACE3.$\mathcal{D}(K, N, A, C)$
$\bar{M} \leftarrow \mathsf{E}^{-1}(K, A\|N, C)$
if $\bar{M}_{n\cdots n+\tau-1} \neq 0^\tau$ then
  return $\perp$
return $\bar{M}_{0\cdots n-1}$
```

Fig. 5: Definition of ACE1, ACE2, ACE3. Here, f is a stream cipher, F is a "tweakable" PRF, E is a tweakable block cipher and 0^τ denotes a string of τ zero bits.

Theorem 5. *Let* $\mathsf{f} : \mathcal{K}_1 \times \mathcal{A} \times \mathcal{N} \to \{0,1\}^n$ *and* $\mathsf{F} : \mathcal{K}_2 \times \mathcal{A} \times \mathcal{N} \times \{0,1\}^n \to \{0,1\}^\tau$ *be two PRFs. Then, for* $\Pi = ACE1[\mathsf{f}, \mathsf{F}]$*, we have*

$$\mathbf{Adv}_{\Pi}^{\mathbf{cae\text{-}a}}(q_e, q_d, q_n, t) \leq \mathbf{Adv}_{\mathsf{f}}^{prf}(q_e+q_d, q_n+q_d, t') + \mathbf{Adv}_{\mathsf{F}}^{prf}(q_e+q_d, q_n+q_d, t'') + \frac{q_d}{2^\tau}$$

if $q_n = 1$*, where* $t' = \alpha(q_e + q_d)$ *and* $t'' = \beta(q_e + q_d)$ *where* α *and* β *are constants dependent on the model of computation.*

The proof of Theorem 5 is a standard hybrid argument followed by a bound on tag-guessing.

Due to the use of stream cipher encryption, ACE1 is not secure if $q_n > 1$, allowing a difference of code fragments to be recovered when (A, N)-pairs repeat. More formally, asking encryption of arbitrary (A, N, M) and (A, N, M') and checking whether the resulting ciphertexts satisfy $C_{\cdots n-1} \oplus C'_{\cdots n-1} = M \oplus M'$ constitutes an effective distinguisher.

One might hope limiting $q_n = 1$ would allow ACE1 to achieve **cae-r** security. However, asking for an encryption of (A, N, M) to obtain $C \| T$, and then querying decryption queries with a varying ciphertext and tag until some (A, N, C') yields M' and checking if $C_{\cdots n-1} \oplus C'_{\cdots n-1} = M \oplus M'$ is again an effective distinguisher. This means that the confidentiality of ACE1 is fully broken after the first forgery. However, ACE1 is secure against reforgeries because ACE1 embodies the "*Independence of* F_{IV} *and* F_T"-paradigm of Forler et. al. [22].

Finally, ACE1 does not resist to targeted forgeries. Given a ciphertext tuple (A, N, C) corresponding to a known plaintext M, an attacker can *deterministically* force the MCU to execute an arbitrary fragment M' by an exhaustive search for the tag of $C_{0\cdots n-1} \oplus M \oplus M'$ with the same (A, N). With a short tag, say $\tau = 8$, an attacker may abuse this to bypass an authentication, for example. Even if the MCU drops the memory decryption key after the first detected forgery, the attacker may simply attack ≈ 256 devices until succeeding, for example to unlock memory protection and extract the device software.

ACE2. ACE2 cannot achieve a non-trivial **cae-r** bound. As the number of decryption queries q_d approaches 2^τ, an effective **cae-r** distinguisher is to search for a tag for some fixed values of $A, N, C_{\cdots n}$, and conclude we interact with **cae-r-id** if and only if the exhaustive tag search fails to yield a forgery. For ACE2, the probability of the latter is 0 (a correct tag always exists), while for a random injection (fixed by (A, N)), the probability is $(2^{n+\tau} - 2^\tau)_{2^\tau} / (2^{n+\tau})_{2^\tau} = (2^\tau(2^n - 1))_{2^\tau} / (2^{n+\tau})_{2^\tau}$, which is close to 1. The **cae-a** security of ACE2 is formalized in Theorem 2.

B Deferred Proofs

Proof (Theorem 1). The analysis is inspired by Theorem 1 by Hoang et. al. [26]. We observe that $\Pr[\mathscr{A}^{\mathbf{cae\text{-}r\text{-}re}_\Pi} \Rightarrow 1] = \Pr[\mathscr{A}^{\mathbf{cae\text{-}a\text{-}re}_\Pi} \Rightarrow 1]$ and then bound $|\Pr[\mathscr{A}^{\mathbf{cae\text{-}r\text{-}id}_\Pi} \Rightarrow 1] - \Pr[\mathscr{A}^{\mathbf{cae\text{-}a\text{-}id}_\Pi} \Rightarrow 1]|$ through a sequence of hybrid games from $\mathbf{cae\text{-}a\text{-}id}_\Pi$ to $\mathbf{cae\text{-}r\text{-}id}_\Pi$. In the first transition, the probability of bad event accounting for possible ciphertext collisions in encryption queries of $\mathbf{cae\text{-}a\text{-}id}_{\Pi}$ is upper bounded. Letting p being the number of unique (A, N) pairs in encryption queries and s_i being the number of encryption queries made with the i^{th} unique (A, N) pair, the bad event probability is upper-bounded as $\sum_{i=1}^{p} s_i(s_i - 1)/2^{n+\tau} \leq q_e(q_n - 1)/2^{n+\tau}$ as $s_i \leq q_n$ and $\sum_{i=1}^{p} s_i = q_e$. In the second transition, the probability of the bad event of a $\mathbf{cae\text{-}r\text{-}id}_{\Pi}$ returning other than $\perp$ is upper-bounded as

$$\sum_{i=1}^{q_d} 2^n/(2^{n+\tau} - q_n + i) \leq \sum_{i=1}^{q_d} 1/(2^\tau - (q_n + q_d)/2^n) \leq q_d/(2^\tau - (q_d + q_n)) \leq 2q_d/2^\tau$$

because the are at most 2^n valid images of the underlying random injection in the i^{th} decryption query made with an (A, N) pair, out of at least $(2^{n+\tau} - q_n + i)$ elements of the co-domain not yet probed by either encryption or decryption queries. The last inequality is due to the assumption $(q_d + q_n) \leq 2^{\tau-1}$.

The proof of the second inequality proceeds similarly as the above, except in the second transition, the probability of the bad event of a $\mathbf{cae\text{-}r\text{-}id}_{\Pi}$ returning other than $\perp$ is now upper-bounded as

$$\sum_{i=1}^{q_d} \frac{|\mathcal{M}'|}{2^{n+\tau} - q_n + i} \leq \sum_{i=1}^{q_d} \frac{|\mathcal{M}'|}{2^n} \cdot \frac{1}{(2^\tau - (q_n + q_d)/2^n)} \leq 2q_d|\mathcal{M}'|/2^{n+\tau}$$

because there are at most $|\mathcal{M}'|$ images of the random injection that are valid and correspond to an element of $\mathcal{M}'$ in the i^{th} decryption query made with an (A, N) pair, out of at least $(2^{n+\tau} - q_n + i)$ elements of the co-domain not yet probed by either encryption or decryption queries, again using the assumption $(q_d + q_n) \leq 2^{\tau-1}$. □

Proof (Lemma 1). We have $\Pr[\mathsf{forge}|f(M_1), \ldots, f(M_\ell), C_1, \ldots C_q] \leq \sum_{i=1}^{q_d} 2^n/(2^{n+\tau} - q - \ell - q_d) \leq 2 \cdot q_d 2^n/2^{n+\tau} = 2q_d/2^\tau$. This is because when testing any one of the strings C_i', there are at least $(2^{n-\tau} - q - \ell - q_d)$ elements of $\{0,1\}^{n+\tau}$ for which it is not yet known whether they are images of f or not and there are never more than 2^n real images. □

Proof (Theorem 5). We start by replacing first f and then F by true random functions $\varphi \leftarrow\$ \text{Func}(\mathcal{A} \times \mathcal{N}, n)$ and $\Phi \leftarrow\$ \text{Func}(\mathcal{A} \times \mathcal{N} \times \{0,1\}^n, \tau)$. We thus have $\mathbf{Adv}^{\mathbf{cae\text{-}a}}_{\text{ACE1}[\mathsf{f},\mathsf{F}]}(q_e, q_d, q_n, t) \leq \mathbf{Adv}^{\text{prf}}_{\mathsf{f}}(q_e + q_d, q_n + q_d, t') + \mathbf{Adv}^{\text{prf}}_{\mathsf{F}}(q_e + q_d, q_n + q_d, t'') + \mathbf{Adv}^{\mathbf{cae\text{-}a}}_{\text{ACE1}[\varphi,\Phi]}(q_e, q_d, q_n)$ by a standard hybrid argument. For the construction $\Pi' = \text{ACE1}[\varphi, \Phi]$, the encryption oracles of the games $\mathbf{cae\text{-}a\text{-}re}_{\Pi'}$ and $\mathbf{cae\text{-}a\text{-}id}_{\Pi'}$ produce identical distributions, so the only way to distinguish the games is a successful forgery in $\mathbf{cae\text{-}a\text{-}re}_{\Pi'}$. This corresponds to guessing a fresh τ-bit image of a random function, which happens with probability no greater than $q_d/2^\tau$. □

Proof (Theorem 2). The proof starts by replacing E and F by $\pi \leftarrow\$ \text{Perm}^{\mathcal{A}\times\mathcal{N}}(n)$ and $\Phi \leftarrow\$ \text{Func}(\mathcal{A} \times \mathcal{N} \times \{0,1\}^n, \tau)$, which yields $\mathbf{Adv}^{\mathbf{cae\text{-}a}}_{\text{ACE2}[\mathsf{E},\mathsf{F}]}(q_e, q_d, q_n, t) \leq$

$\mathbf{Adv}^{\pm\widetilde{\mathrm{prp}}}_{\mathsf{E}}(q_e+q_d,t')+\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(q_e+q_d,t'')+\mathbf{Adv}^{\mathbf{cae\text{-}a}}_{\mathrm{ACE2}[\pi,\Phi]}(q_e,q_d,q_n)$ by a standard hybrid argument. The analysis of $\mathbf{Adv}^{\mathbf{cae\text{-}a}}_{\mathrm{ACE2}[\pi,\Phi]}(q_e,q_d,q_n)$ starts by replacing π by a collection of random functions φ with the same signature, which means an RP-RF switch for every unique (A,N) pair that appears in encryption queries. With p being the number of such unique pairs and s_i being the number of encryption queries made with the i^{th} unique (A,N) pair, the resulting change in advantage is upper-bounded as $\sum_{i=1}^{p} s_i(s_i-1)/2^n \leq q_e(q_n-1)/2^n$ as $s_i \leq q_n$ and $\sum_{i=1}^{p} s_i = q_e$. Then, the only way to distinguish ACE2$[\varphi,\Phi]$ is through a successful forgery, which happens with probability no greater than $q_d/2^{\tau}$, Φ being a random function.

The proof of the **cae-a**$[\mathcal{M}']$ bound proceeds similarly as the above, except that we first account for the probability of a forgery against $\Pi' = \mathrm{ACE2}[\pi,\Phi]$ and only then replace π with a collection of random functions φ. Formally, this is done through an introduction of a hybrid game G that is identical to **cae-r-re**, except that its decryption oracle always rejects. We have $|\Pr[\mathscr{A}^{\mathbf{cae\text{-}a\text{-}re}_{\Pi'}} \Rightarrow 1] - \Pr[\mathscr{A}^{\mathbf{cae\text{-}a\text{-}id}_{\Pi'}} \Rightarrow 1]| \leq |\Pr[\mathscr{A}^{\mathbf{cae\text{-}r\text{-}re}_{\Pi'}} \Rightarrow 1] - \Pr[\mathscr{A}^{G} \Rightarrow 1]| + |\Pr[\mathscr{A}^{G} \Rightarrow 1] - \Pr[\mathscr{A}^{\mathbf{cae\text{-}a\text{-}id}_{\Pi'}} \Rightarrow 1]|$. In the first transition, a successful forgery (A,N,C) with $\bar{C} = C_{0\cdots n-1}$ and $T = C_{n\cdots n+\tau-1}$ must have a valid tag T and additionally, the preimage $\pi^{-1}(A,N,\bar{C})$ must fall in $\mathcal{M}'$. The probability of the former is no more than $1/2^{-\tau}$. The probability of the latter is no more than $|\mathcal{M}'|/(2^n-q_n) \leq 2|\mathcal{M}'|/2^n$, as prior to a forgery, there are at least (2^n-q_n) elements of $\{0,1\}^n$ whose preimage under $\pi(A,N,\cdot)$ is not yet known and using $q_n \leq 2^{n-1}$. The analysis of the second transition, replacemnt of π with φ, is the same as for the **cae-a** analysis. □

C Algorithmic Descriptions and Test Vectors for ChiLow

The complete algorithmic descriptions of ChiLow-$(32+\tau)$ and ChiLow-40 can be found in Algorithm 1 and Algorithm 2, respectively. The test vectors of ChiLow-$(32+\tau)$ and ChiLow-40 in hexadecimal format, along with the intermediate values after each round, are provided in Table 6 and Table 7, respectively.

Algorithm 1: ChiLow-$(32+\tau)$ Decryption

```
Data: 128-bit key $\mathsf{K}$, 64-bit tweak $\mathsf{T}$, 32-bit ciphertext $\mathsf{C}$, $\tau$-bit $\mathsf{GTag}$, number
      of rounds $\mathsf{rnd}$
Result: 32-bit plaintext $\mathsf{P}$ or $\perp$

$\mathsf{X} \leftarrow \mathsf{C} \oplus \mathsf{K}_{64\cdots95}$                      ▷ whitening cipher state
$\mathsf{X'} \leftarrow \mathsf{C} \oplus \mathsf{K}_{96\cdots127}$                    ▷ whitening tag state
$\mathsf{T} \leftarrow \mathsf{T} \oplus \mathsf{K}_{0\cdots63}$                       ▷ whitening tweak state

for $i = 0$ to $rnd - 2$ do
    $\mathsf{K}_{96\cdots127} \leftarrow \mathsf{K}_{96\cdots126} \oplus c^{(i)}$      ▷ adding round constant

    $\mathsf{X} \leftarrow \chi_{32}(\mathsf{X})$                                     ▷ the non-linear layer
    $\mathsf{X'} \leftarrow \chi_{32}(\mathsf{X'})$
    $\mathsf{T} \leftarrow \chi_{64}(\mathsf{T})$
    $\mathsf{K} \leftarrow \chi_{128}(\mathsf{K})$

    $\mathsf{X} \leftarrow L_{32}(\mathsf{X})$                                        ▷ the linear layer
    $\mathsf{X'} \leftarrow L'_{32}(\mathsf{X'})$
    $\mathsf{T} \leftarrow L_{64}(\mathsf{T})$
    $\mathsf{K} \leftarrow L_{128}(\mathsf{K})$

    $\mathsf{X} \leftarrow \mathsf{X} \oplus \mathsf{T}_{0\cdots31}$                  ▷ interaction from tweak state to cipher state
    $\mathsf{X'} \leftarrow \mathsf{X'} \oplus \mathsf{T}_{32\cdots63}$               ▷ interaction from tweak state to tag state
    $\mathsf{T} \leftarrow \mathsf{T} \oplus \mathsf{K}_{0\cdots63}$                  ▷ interaction from key state to tweak state

$\mathsf{X} \leftarrow \chi_{32}(\mathsf{X})$                                         ▷ the last round
$\mathsf{X'} \leftarrow \chi_{32}(\mathsf{X'})$
$\mathsf{T} \leftarrow L_{64}(\mathsf{T})$
$\mathsf{P} \leftarrow \mathsf{X} \oplus \mathsf{T}_{0\cdots31}$
$\mathsf{Tag} \leftarrow (\mathsf{X'} \oplus \mathsf{T}_{32\cdots63})_{0\cdots\tau-1}$

if $Tag = GTag$ then
    return $P$                                                                         ▷ authentication

return $\perp$
```

Table 6: Test vector of ChiLow-$(32+\tau)$ in hexadecimal.

	X	X'	T	K
Input	0x01234567		0x0011223344556677	0xFEDCBA98765432107766554433221100
Whitening	0x77777777	0xFFFFFFFF	0x7777777777777777	0xFEDCBA98765432107766554433221100
Round 0	0x28EAF2BE	0xCC444CCC	0x8A31D5C4C3D916A3	0x80863D49277069D7398262F7F0EA3580
Round 1	0x74C91B66	0x82D61EF2	0xE7D3C7CC8E2128C3	0xBA1137AD4E9540169B62135092FAF499
Round 2	0x0CB760FD	0x9F9EC860	0x2F1334F1CE208EF1	0x4660767CD7CD20EEAFCADEB29D06F484
Round 3	0x941BE07F	0x2A6B6FEC	0x07F94B5D8D2AE616	0x54357497D3625CF971F8FE86980DE74E
Round 4	0x537CCC17	0x329FD83B	0x057CD8A34158650C	0x38A9B2E9D3433275E0202F68DC296C2A
Round 5	0x0A710EDE	0xB56E008C	0x173795E888B71A95	0x861B7DCBF0A1F4CC9DF37134EDCE1460
Round 6	0x4AD82C9B	0x3D3A188F	0xA20CA67F46A99234	0x01234D7253D8186C3C17BB32A6ECF192
Round 7	0x2E75D127	0x0FBC7E64	0x33C4C4CB763F749E	0x01234D7253D8186C3C17BB32A6ECF192

Algorithm 2: ChiLow-40 Decryption

```
Data: 128-bit key $\mathsf{K}$, 64-bit tweak $\mathsf{T}$, 40-bit ciphertext $\mathsf{C}$, number of rounds $\mathsf{rnd}$
Result: 32-bit plaintext $\mathsf{P}$ or $\perp$

$\mathsf{X} \leftarrow \mathsf{C} \oplus \mathsf{K}_{64\cdots103}$                ▷ whitening cipher state
$\mathsf{T} \leftarrow \mathsf{T} \oplus \mathsf{K}_{0\cdots63}$                ▷ whitening tweak state

for $i = 0$ to $rnd - 2$ do
    $\mathsf{K}_{96\cdots127} \leftarrow \mathsf{K}_{96\cdots126} \oplus c^{(i)}$     ▷ adding round constant
    $\mathsf{X} \leftarrow \chi_{40}(\mathsf{X})$                      ▷ the non-linear layer
    $\mathsf{T} \leftarrow \chi_{64}(\mathsf{T})$
    $\mathsf{K} \leftarrow \chi_{128}(\mathsf{K})$
    $\mathsf{X} \leftarrow L_{40}(\mathsf{X})$                      ▷ the linear layer
    $\mathsf{T} \leftarrow L_{64}(\mathsf{T})$
    $\mathsf{K} \leftarrow L_{128}(\mathsf{K})$
    $\mathsf{X} \leftarrow \mathsf{X} \oplus \mathsf{T}_{0\cdots39}$        ▷ interaction from tweak state to cipher state
    $\mathsf{T} \leftarrow \mathsf{T} \oplus \mathsf{K}_{0\cdots63}$        ▷ interaction from key state to tweak state

$\mathsf{X} \leftarrow \chi_{40}(\mathsf{X})$                          ▷ the last round
$\mathsf{T} \leftarrow L_{64}(\mathsf{T})$
$\mathsf{X} \leftarrow \mathsf{X} \oplus \mathsf{T}_{0\cdots39}$
$\mathsf{P} \leftarrow \mathsf{X}_{0\ldots31}$
$\mathsf{Tag} \leftarrow \mathsf{X}_{32\ldots39}$
if $Tag = 0^8$ then
    return $P$                                  ▷ authentication
return $\perp$
```

Table 7: Test vector of ChiLow-40 in hexadecimal.

Round	X	T	K
Input	0x317C83E4A7	0x0011223344556677	0xFEDCBA98765432107766554433221100
Whitening	0xA90AD7D6B7	0x7777777777777777	0xFEDCBA98765432107766554433221100
Round 0	0xDFBD12AE8A	0x8A21D5C4C3D916A3	0x81863D4B277069D7399262F7F0EA3580
Round 1	0x791D013E2A	0xEB4BCF84EC3329C2	0x335137BD6F9144549FF21358B0AAF499
Round 2	0xC746DFAA6C	0xB58F74ADF0D402F6	0xDA502A4666796686877DDA74A49F8CD4
Round 3	0x367A387031	0x388188393672AFDB	0x978C2DD89FE41AF5E3D1D0FDA206B258
Round 4	0x9C441248D5	0x6C2AE03090DDD441	0x0E76D7859536F033756F425E3DD27CEF
Round 5	0xBDA31753FF	0x01AC33739BE14EAE	0xFF7E2DD4055C0351A626C0A79B1A8A29
Round 6	0xAE828033F8	0xB22148B5EA79366C	0x46413C9DEA941772F2BB69807E27E0D2
Round 7	0x0090545706	0x55952186B27460FC	0x46413C9DEA941772F2BB69807E27E0D2

D Proof of Theorem 4: Invertivibility of ChiChi

Before we study each formula, we proof a helpful lemma.

Lemma 2. *For $y = \chi_m(x)$, $l \in \{0,1\}$ and $l \le j \le i-l$, we have*

$$y_{2i-l} \prod_{k=j}^{i-l} \overline{y}_{2k-1+l} = y_{2i-l} \prod_{k=j}^{i-l} \overline{x}_{2k-1+l}.$$

Proof. We proof the statement by induction over j. For the base case, consider $j = i-l$. The statement clearly holds as

$$\begin{aligned}
y_{2i-l}\overline{y}_{2(i-l)-1+l} &= y_{2i-l}\overline{y}_{2i-1-l} \\
&= (x_{2i-l} + \overline{x}_{2i+1-l}x_{2i+2-l})(\overline{x}_{2i-1-l} + \overline{x}_{2i-l}x_{2i+1-l}) \\
&= x_{2i-l}\overline{x}_{2i-1-l} + \overline{x}_{2i+1-l}x_{2i+2-l}\overline{x}_{2i-1-l} \\
&= (x_{2i-l} + \overline{x}_{2i+1-l}x_{2i+2-l})\overline{x}_{2i-1-l} \\
&= y_{2i-l}\overline{x}_{2i-1-l} = y_{2i-l}\overline{x}_{2(i-l)-1+l}.
\end{aligned}$$

Now, for the step $j \to j-1$, where $l < j \le i$, we have

$$\begin{aligned}
y_{2i-l} \prod_{k=j-1}^{i-l} \overline{y}_{2k-1+l} &= \left(y_{2i-l} \prod_{k=j}^{i-l} \overline{y}_{2k-1+l} \right) \overline{y}_{2j-3+l} \\
&= \left(y_{2i-l} \prod_{k=j}^{i-l} \overline{x}_{2k-1+l} \right) (\overline{x}_{2j-3+l} + \overline{x}_{2j-2+l}x_{2j-1+l}) \\
&= y_{2i-l} \prod_{k=j-1}^{i-l} \overline{x}_{2k-1+l}.
\end{aligned}$$

□

Lemma 3. *Formula for computing x_m. For m even and $y = \chi_{2m}(x)$, we have*

$$x_m = y_{m-3} + \left(\sum_{i=0}^{\frac{m}{2}-2} y_{2i} \prod_{k=1}^{i} \overline{y}_{2k-1} \right) \cdot \left(\overline{y}_m + \left(\sum_{i=1}^{\frac{m}{2}-1} y_{2i+m} \prod_{k=1}^{i} \overline{y}_{2k+m-1} \right) + y_{m-2} \prod_{k=1}^{\frac{m}{2}} \overline{y}_{2k+m-1} \right).$$

Proof. The proof is straightforward, in the sense that we plugin the definition of the y_i and then simplify the terms. Recall that $y_{m-3} = x_m + \overline{x}_{m-2}x_0$. Hence, we show that the product simplifies to $\overline{x}_{m-2}x_0$. For this, we show that the left-hand factor equals

$$x_0 + \overline{x}_1 x_{m-2} \prod_{k=1}^{\frac{m}{2}-2} \overline{x}_{2k+1}$$

and the right-hand factor equals

$$\overline{x}_{m-2} + \overline{x}_0 x_1 \prod_{k=1}^{\frac{m}{2}} \overline{y}_{2k+m-1}.$$

Then, the product of those two clearly evaluates to $x_0\overline{x}_{m-2}$ and hence cancels the same term in the definition of y_{m-3} leaving only x_m, as desired.

The left-hand factor. As we are considering index values of y which are smaller than $m-3$, we do not need to consider the linear feed-forward. Hence, we can apply Lemma 2 and plugin in the definition of the $y_i = x_i + \overline{x}_{i+1}x_{i+2}$ to get

$$\begin{aligned}\sum_{i=0}^{\frac{m}{2}-2} y_{2i} \prod_{k=1}^{i} \overline{y}_{2k-1} &= \sum_{i=0}^{\frac{m}{2}-2} y_{2i} \prod_{k=1}^{i} \overline{x}_{2k-1} \\ &= \sum_{i=0}^{\frac{m}{2}-2} (x_{2i} + \overline{x}_{2i+1}x_{2i+2}) \prod_{k=1}^{i} \overline{x}_{2k-1}.\end{aligned}$$

Now, simple reordering yields

$$\begin{aligned}\sum_{i=0}^{\frac{m}{2}-2} y_{2i} \prod_{k=1}^{i} \overline{y}_{2k-1} &= \sum_{i=0}^{\frac{m}{2}-2} \left(x_{2i} \prod_{k=1}^{i} \overline{x}_{2k-1} + \overline{x}_{2i+1}x_{2i+2} \prod_{k=1}^{i} \overline{x}_{2k-1} \right) \\ &= \sum_{i=0}^{\frac{m}{2}-2} \left(x_{2i} \prod_{k=1}^{i} \overline{x}_{2k-1} + x_{2i+2} \prod_{k=1}^{i+1} \overline{x}_{2k-1} \right).\end{aligned}$$

This is a telescope sum, i.e., the right-hand side of the i-th summand cancels the left-hand side of the $(i+1)$-th sumand such that we are left with only the left-hand side of the first and the right-hand side of the last summand respectively. In other words,

$$\sum_{i=0}^{\frac{m}{2}-2} y_{2i} \prod_{k=1}^{i} \overline{y}_{2k-1} = x_0 + x_{m-2} \prod_{k=1}^{\frac{m}{2}-1} \overline{x}_{2k-1} = x_0 + \overline{x}_1 x_{m-2} \prod_{k=1}^{\frac{m}{2}-2} \overline{x}_{2k+1}$$

which constitutes the desired form.

The right-hand factor. Recall that for the right-hand factor we want to show that

$$\overline{y}_m + \left(\sum_{i=1}^{\frac{m}{2}-1} y_{2i+m} \prod_{k=1}^{i} \overline{y}_{2k+m-1} \right) + y_{m-2} \prod_{k=1}^{\frac{m}{2}} \overline{y}_{2k+m-1} \overset{!}{=} \overline{x}_{m-2} + \overline{x}_0 x_1 \prod_{k=1}^{\frac{m}{2}} \overline{y}_{2k+m-1}.$$

First, we move the last summand of the middle sum out of the sum. Thereby, all the indexes use in the remaining sum do not require any special attention and we can apply exatcly the same techniques as for the left-hand factor. Hence,

$$\begin{aligned}
&\overline{y}_m + \left(\sum_{i=1}^{\frac{m}{2}-1} y_{2i+m} \prod_{k=1}^{i} \overline{y}_{2k+m-1}\right) + y_{m-2} \prod_{k=1}^{\frac{m}{2}} \overline{y}_{2k+m-1} \\
&= \overline{y}_m + \left(\sum_{i=1}^{\frac{m}{2}-2} y_{2i+m} \prod_{k=1}^{i} \overline{y}_{2k+m-1}\right) + y_{2m-2} \prod_{k=1}^{\frac{m}{2}-1} \overline{y}_{2k+m-1} + y_{m-2} \prod_{k=1}^{\frac{m}{2}} \overline{y}_{2k+m-1} \\
&= \overline{y}_m + \left(\sum_{i=1}^{\frac{m}{2}-2} y_{2i+m} \prod_{k=1}^{i} \overline{x}_{2k+m-1}\right) + y_{2m-2} \prod_{k=1}^{\frac{m}{2}-1} \overline{y}_{2k+m-1} + y_{m-2} \prod_{k=1}^{\frac{m}{2}} \overline{y}_{2k+m-1} \\
&= \overline{y}_m + \left(\overline{x}_{m+1}x_{m+2} + x_{2m-2} \prod_{k=1}^{\frac{m}{2}-1} \overline{x}_{2k+m-1}\right) + y_{2m-2} \prod_{k=1}^{\frac{m}{2}-1} \overline{y}_{2k+m-1} + y_{m-2} \prod_{k=1}^{\frac{m}{2}} \overline{y}_{2k+m-1}.
\end{aligned}$$

Next, consider the relevant *special* cases of the definition of $y = \chi_{2m}(x)$, namely

$$\begin{aligned}
y_{m-3} &= x_m + \overline{x}_{m-2}x_0 \\
y_{m-2} &= x_{m-1} + \overline{x}_0 x_1 \\
y_m &= x_{m-2} + \overline{x}_{m+1}x_{m+2} \\
y_{2m-2} &= x_{2m-2} + \overline{x}_{2m-1}x_{m-1} \\
y_{2m-1} &= x_{2m-1} + \overline{x}_{m-1}x_m.
\end{aligned}$$

Plugging in y_m, then y_{2m-2} and finally y_{m-2} yields

$$\begin{aligned}
&\overline{x}_{m-2} + x_{2m-2} \prod_{k=1}^{\frac{m}{2}-1} \overline{x}_{2k+m-1} + y_{2m-2} \prod_{k=1}^{\frac{m}{2}-1} \overline{y}_{2k+m-1} + y_{m-2} \prod_{k=1}^{\frac{m}{2}} \overline{y}_{2k+m-1} \\
&= \overline{x}_{m-2} + \overline{x}_{2m-1}x_{m-1} \prod_{k=1}^{\frac{m}{2}-1} \overline{y}_{2k+m-1} + y_{m-2} \prod_{k=1}^{\frac{m}{2}} \overline{y}_{2k+m-1} \\
&= \overline{x}_{m-2} + x_{m-1} \prod_{k=1}^{\frac{m}{2}} \overline{y}_{2k+m-1} + y_{m-2} \prod_{k=1}^{\frac{m}{2}} \overline{y}_{2k+m-1} \\
&= \overline{x}_{m-2} + \overline{x}_0 x_1 \prod_{k=1}^{\frac{m}{2}} \overline{y}_{2k+m-1}
\end{aligned}$$

which is the desired form. This concludes the correctness proof of the formula for x_m. □

Lemma 4. *Formula for computing $x_m + x_{m-3}$. For m even and $y = \chi_{2m}(x)$, we have*

$$x_m + x_{m-3} = y_{m-1} + \left(\overline{y}_{m-3} + \overline{y}_m \sum_{i=0}^{\frac{m}{2}-2} y_{2i} \prod_{k=1}^{i} \overline{y}_{2k-1}\right) \cdot \left(\sum_{i=\frac{m}{2}+1}^{m} y_{2i-1} \prod_{k=\frac{m}{2}+1}^{i-1} \overline{y}_{2k}\right).$$

Proof. As $y_{m-1} = x_{m-3} + x_m + \overline{x}_m x_{m+1}$, we have to show that the product equals $\overline{x}_m x_{m+1}$. For this, we show that the left-hand factor equals

$$\overline{x}_m + \overline{x}_{m+1} x_{m+2} \left(x_0 + \overline{x}_1 x_{m-2} \prod_{k=1}^{\frac{m}{2}-2} \overline{x}_{2k+1} \right)$$

and the right-hand factor equals

$$x_{m+1} + x_m \overline{x}_{m+2} \prod_{k=\frac{m}{2}+2}^{m} \overline{x}_{2k}.$$

Then, the product clearly evaluates to $\overline{x}_m x_{m+1}$.

The left-hand factor. From the proof of Lemma 3, we already know that

$$\sum_{i=0}^{\frac{m}{2}-2} y_{2i} \prod_{k=1}^{i} \overline{y}_{2k-1} = x_0 + \overline{x}_1 x_{m-2} \prod_{k=1}^{\frac{m}{2}-2} \overline{x}_{2k+1}.$$

Multiplying this with $\overline{y}_m = \overline{x}_{m-2} + \overline{x}_{m+1} x_{m+2}$ gives

$$\overline{x}_{m-2} x_0 + \overline{x}_{m+1} x_{m+2} \left(x_0 + \overline{x}_1 x_{m-2} \prod_{k=1}^{\frac{m}{2}-2} \overline{x}_{2k+1} \right)$$

as the term containing $\overline{x}_{m-2} x_{m-2}$ vanishes. Finally, we add $\overline{y}_{m-3} = \overline{x}_m + \overline{x}_{m-2} x_0$ and, as $\overline{x}_{m-2} x_0$ is canceled, are left with

$$\overline{x}_m + \overline{x}_{m+1} x_{m+2} \left(x_0 + \overline{x}_1 x_{m-2} \prod_{k=1}^{\frac{m}{2}-2} \overline{x}_{2k+1} \right)$$

which is the desired form.

The right-hand factor. For the right-hand factor

$$\sum_{i=\frac{m}{2}+1}^{m} y_{2i-1} \prod_{k=\frac{m}{2}+1}^{i-1} \overline{y}_{2k},$$

observe that all y are from the output of χ_{m+1}. Thus, we first substitute $x'_i = x_{i+(m-1)}$ and likewise $y'_i = y_{i+(m-1)}$. With this, we have

$$\begin{aligned} \sum_{i=\frac{m}{2}+1}^{m} y_{2i-1} \prod_{k=\frac{m}{2}+1}^{i-1} \overline{y}_{2k} &= \sum_{i=\frac{m}{2}+1}^{m} y'_{2i-1-(m-1)} \prod_{k=\frac{m}{2}+1}^{i-1} \overline{y'}_{2k-(m-1)} \\ &= \sum_{i=\frac{m}{2}+1}^{m} y'_{2(i-\frac{m}{2})} \prod_{k=\frac{m}{2}+1}^{i-1} \overline{y'}_{2(k-\frac{m}{2})+1} \\ &= \sum_{i=1}^{\frac{m}{2}} y'_{2i} \prod_{k=1}^{i-1} \overline{y'}_{2k+1} = \sum_{i=1}^{\frac{m}{2}} y'_{2i} \prod_{k=2}^{i} \overline{y'}_{2k-1}. \end{aligned}$$

Now, by Lemma 2, we can simply replace $\overline{y'}$ by $\overline{x'}$ in the product. Then, we again get a telescope sum and finally the desired form by reversing the substitution. That is,

$$\begin{aligned}
\sum_{i=\frac{m}{2}+1}^{m} y_{2i-1} \prod_{k=\frac{m}{2}+1}^{i-1} \overline{y}_{2k} &= \sum_{i=1}^{\frac{m}{2}} y'_{2i} \prod_{k=2}^{i} \overline{x'}_{2k-1} \\
&= \sum_{i=1}^{\frac{m}{2}} \left(x'_{2i} \prod_{k=2}^{i} \overline{x'}_{2k-1} + x'_{2i} \prod_{k=2}^{i+1} \overline{x'}_{2k-1} \right) \\
&= x'_2 + x'_{m+2} \prod_{k=2}^{\frac{m}{2}+1} \overline{x'}_{2k-1} = x'_2 + x'_1 \prod_{k=2}^{\frac{m}{2}+1} \overline{x'}_{2k-1} \\
&= x_{m+1} + x_m \prod_{k=2}^{\frac{m}{2}+1} \overline{x}_{2k-1+(m-1)} \\
&= x_{m+1} + x_m \prod_{k=2}^{\frac{m}{2}+1} \overline{x}_{2(k-1+\frac{m}{2})} \\
&= x_{m+1} + x_m \prod_{k=\frac{m}{2}+1}^{m} \overline{x}_{2k} \\
&= x_{m+1} + x_m \overline{x}_{m+2} \prod_{k=\frac{m}{2}+2}^{m} \overline{x}_{2k}.
\end{aligned}$$

□

Lemma 5. *Formula for computing x_{m-1}. For m even and $y = \chi_{2m}(x)$, we have*

$$\begin{aligned}
x_{m-1} = y_{m-2} &+ \left(\sum_{i=1}^{\frac{m}{2}-2} y_{2i-1} \prod_{k=0}^{i-1} \overline{y}_{2k} \right) \\
&+ \left(\overline{y}_{m-1} + \left(\overline{y}_{m+1} + \sum_{i=\frac{m}{2}+2}^{m} y_{2i-1} \prod_{k=\frac{m}{2}+1}^{i-1} \overline{y}_{2k} \right) \overline{y}_{m-3} \right) \prod_{k=0}^{\frac{m}{2}-2} \overline{y}_{2k}.
\end{aligned}$$

Proof. As $y_{m-2} = x_{m-1} + \overline{x}_0 x_1$, we show that the rest of the sum evaluates to $\overline{x}_0 x_1$. To do so, we again apply Lemma 2 to the second summand and then simplify the telescope sum. That is,

$$\begin{aligned}\sum_{i=1}^{\frac{m}{2}-2} y_{2i-1}\prod_{k=0}^{i-1}\overline{y}_{2k} &= \sum_{i=1}^{\frac{m}{2}-2} y_{2i-1}\prod_{k=0}^{i-1}\overline{x}_{2k}\\ &= \sum_{i=1}^{\frac{m}{2}-2}\left(x_{2i-1}\prod_{k=0}^{i-1}\overline{x}_{2k} + x_{2i+1}\prod_{k=0}^{i}\overline{x}_{2k}\right)\\ &= \overline{x}_0 x_1 + x_{m-3}\prod_{k=0}^{\frac{m}{2}-2}\overline{x}_{2k}.\end{aligned}$$

Hence, we have to show that the third summand evaluates to $x_{m-3}\prod_{k=0}^{\frac{m}{2}-2}\overline{x}_{2k}$. For the inner part of the third summand, using the same techniques as before and slightly abusing notation such that $x_{2m} = x_{m-1}$ and $x_{2m+1} = x_m$, we have

$$\begin{aligned}\sum_{i=\frac{m}{2}+2}^{m} y_{2i-1}\prod_{k=\frac{m}{2}+1}^{i-1}\overline{y}_{2k} &= \sum_{i=\frac{m}{2}+2}^{m} y_{2i-1}\prod_{k=\frac{m}{2}+1}^{i-1}\overline{x}_{2k}\\ &= \sum_{i=\frac{m}{2}+2}^{m}\left(x_{2i-1}\prod_{k=\frac{m}{2}+1}^{i-1}\overline{x}_{2k} + x_{2i+1}\prod_{k=\frac{m}{2}+1}^{i}\overline{x}_{2k}\right)\\ &= \overline{x}_{m+2}x_{m+3} + \overline{x}_{2m+1}\prod_{k=\frac{m}{2}+1}^{m}\overline{x}_{2k}\\ &= \overline{x}_{m+2}x_{m+3} + \overline{x}_{m-1}x_m\prod_{k=\frac{m}{2}+1}^{m-1}\overline{x}_{2k}.\end{aligned}$$

Next, we add $\overline{y}_{m+1} = \overline{x}_{m+1} + \overline{x}_{m+2}x_{m+3}$ to get

$$\overline{x}_{m+1} + \overline{x}_{m-1}x_m\prod_{k=\frac{m}{2}+1}^{m-1}\overline{x}_{2k}.$$

Now, we multiply with $\overline{y}_{m-3} = \overline{x}_m + \overline{x}_{m-2}x_0$ and get

$$\overline{x}_m\overline{x}_{m+1} + x_0\overline{x}_{m-2}\left(\overline{x}_{m+1} + \overline{x}_{m-1}x_m\prod_{k=\frac{m}{2}+1}^{m-1}\overline{x}_{2k}\right).$$

Then, we add $\overline{y}_{m-1} = x_{m-3} + \overline{x}_m + \overline{x}_m x_{m+1}$. For this, first observe that

$$\overline{x}_m\overline{x}_{m+1} + \overline{x}_m + \overline{x}_m x_{m+1} = \overline{x}_m(\overline{x}_{m+1} + 1 + x_{m+1}) = 0$$

and hence we are left with

$$x_{m-3} + x_0\overline{x}_{m-2}\left(\overline{x}_{m+1} + \overline{x}_{m-1}x_m\prod_{k=\frac{m}{2}+1}^{m-1}\overline{x}_{2k}\right).$$

Finally, we multiply with $\prod_{k=0}^{\frac{m}{2}-2} \overline{y}_{2k}$. For the second summand, we get a telescope product, i.e.,

$$x_0(\overline{x}_0 + \overline{x}_1 x_2)(\overline{x}_2 + \overline{x}_3 x_4) \cdot \ldots \cdot (\overline{x}_{m-4} + \overline{x}_{m-3} x_{m-2})\overline{x}_{m-2} = 0$$

and hence we are only left with

$$x_{m-3} \prod_{k=0}^{\frac{m}{2}-2} \overline{y}_{2k}.$$

With the same technique as given in the proof of Lemma 2, we can swith $\overline{y}$ to $\overline{x}$ and obtain

$$x_{m-3} \prod_{k=0}^{\frac{m}{2}-2} \overline{x}_{2k}.$$

This concludes the proof as this is exactly the term we wanted to obtain. □

Lemma 6. *Formula for computing x_{m-2}. For m even and $y = \chi_{2m}(x)$, we have*

$$\begin{aligned} x_{m-2} = y_m &+ \left(\sum_{i=\frac{m}{2}+1}^{m-1} y_{2i} \prod_{k=\frac{m}{2}+1}^{i} \overline{y}_{2k-1} \right) \\ &+ \left(y_{m-2} + \left(\sum_{i=1}^{\frac{m}{2}-1} y_{2i-1} \prod_{k=0}^{i-1} \overline{y}_{2k} \right) + y_{m-1} \prod_{k=0}^{\frac{m}{2}-2} \overline{y}_{2k} \right) \prod_{k=\frac{m}{2}+1}^{m} \overline{y}_{2k-1}. \end{aligned}$$

Proof. We have $y_m = x_{m-2} + \overline{x}_{m+1} x_{m+2}$ and for the second summand, with the same techniques as before, we get

$$\begin{aligned} \sum_{i=\frac{m}{2}+1}^{m-1} y_{2i} \prod_{k=\frac{m}{2}+1}^{i} \overline{y}_{2k-1} &= \sum_{i=\frac{m}{2}+1}^{m-1} y_{2i} \prod_{k=\frac{m}{2}+1}^{i} \overline{x}_{2k-1} \\ &= \sum_{i=\frac{m}{2}+1}^{m-1} \left(x_{2i} \prod_{k=\frac{m}{2}+1}^{i} \overline{x}_{2k-1} + x_{2i+2} \prod_{k=\frac{m}{2}+1}^{i+1} \overline{x}_{2k-1} \right) \\ &= \overline{x}_{m+1} x_{m+2} + x_{2m} \prod_{k=\frac{m}{2}+1}^{m} \overline{x}_{2k-1} \\ &= \overline{x}_{m+1} x_{m+2} + x_{m-1} \prod_{k=\frac{m}{2}+1}^{m} \overline{x}_{2k-1} \end{aligned}$$

and hence have to show that the third summand evaluates to

$$x_{m-1} \prod_{k=\frac{m}{2}+1}^{m} \overline{x}_{2k-1}.$$

For the third summand, we first move the summand with index $i = \frac{m}{2} - 1$ out such that we can apply Lemma 2. That is,

$$\begin{aligned}\sum_{i=1}^{\frac{m}{2}-2} y_{2i-1} \prod_{k=0}^{i-1} \overline{y}_{2k} &= \sum_{i=1}^{\frac{m}{2}-2} y_{2i-1} \prod_{k=0}^{i-1} \overline{x}_{2k} \\ &= \sum_{i=1}^{\frac{m}{2}-2} \left(x_{2i-1} \prod_{k=0}^{i-1} \overline{x}_{2k} + x_{2i+1} \prod_{k=0}^{i} \overline{x}_{2k} \right) \\ &= \overline{x}_0 x_1 + x_{m-3} \prod_{k=0}^{\frac{m}{2}-2} \overline{x}_{2k}.\end{aligned}$$

For the moved term, we have

$$y_{m-3} \prod_{k=0}^{\frac{m}{2}-2} \overline{y}_{2k} = (x_m + \overline{x}_{m-2} x_0) \prod_{k=0}^{\frac{m}{2}-2} (\overline{x}_{2k} + \overline{x}_{2k+1} x_{2k+2})$$

which again contains a telescope product and hence simplifies to

$$x_m \prod_{k=0}^{\frac{m}{2}-2} \overline{y}_{2k}.$$

Adding the two terms above and $y_{m-2} = x_{m-1} + \overline{x}_0 x_1$ yields

$$x_{m-1} + x_{m-3} \prod_{k=0}^{\frac{m}{2}-2} \overline{x}_{2k} + x_m \prod_{k=}^{\frac{m}{2}-2} \overline{y}_{2k}.$$

Next, we add the term

$$y_{m-1} \prod_{k=0}^{\frac{m}{2}-2} \overline{y}_{2k}$$

where $y_{m-1} = x_{m-3} + x_m + \overline{x}_m x_{m+1}$. Therefore, the terms with the leading x_m cancel and we can apply the same technique as in Lemma 2 to see that the terms with the leading x_{m-3} also cancels. Hence, we get

$$x_{m-1} + \overline{x}_m x_{m+1} \prod_{k=0}^{\frac{m}{2}-2} \overline{y}_{2k}.$$

Finally, we multiply with $\prod_{k=\frac{m}{2}+1}^{m} \overline{y}_{2k-1}$. By once again applying the techniques from Lemma 2 and on the other hand using that the telescope product vanishes, we obtain

$$x_{m-1} \prod_{k=\frac{m}{2}+1}^{m} \overline{x}_{2k-1}$$

which is as desired and hence concludes the proof. □

E Differential Trails

E.1 Differential Properties of $\chi\!\!\chi$

Extended affine equivalence of χ variants. As we use variants of two concatenated χ functions as non-linear mapping in our different round functions, we would like to use all the properties of χ. As any variant of $\chi\!\!\chi$ where the linear part is changed is affine equivalent to two concatenated χ, differential probability and squared linear correlation over $\chi\!\!\chi$ correspond to the multiplication of the differential probability or squared linear correlation for each χ.

Therefore, we can use the properties from [34] and [35] to compute differential probability and linear correlation and model efficiently differential and linear properties of ChiLow in MILP.

E.2 MILP Modeling for Differential Trails

Due to the nature of the non-linear layer $\chi\!\!\chi$, we made bit-wise modeling of the cipher. For the linear layer, the model consists of 8 equations per bit to model the 3-bit `XOR` and later improve to get rid of the dummy variable. For the non-linear layer, the modeling requires a bit more attention. We first model the propagation through the `AND` with one equation representing the only impossible transition from two input (x) to one output bit (y):

$$x_i + x_{i+1} - y_i \geq 0\,.$$

Doing so, we consider local propagation only for each output bit, however, this is not enough as some impossible transition can happen when we look at two output bits depending on three input bits. This case can be excluded with the following equations:

$$\begin{aligned} x_i - x_{i+1} + x_{i+2} + y_i - y_{i+1} &< 3 \\ x_i - x_{i+1} + x_{i+2} - y_i + y_{i+1} &< 3 \end{aligned}$$

Then we can model the linear part of $\chi\!\!\chi$ using 4 equations for almost all positions, and 8 for the one where the output is the sum of three bits.

This modeling works for χ applied in big circles as in Subterranean [17] or Koala [21], but in the case of ChiLow, the division of the state into two χ-like functions leads to considering impossible transitions where the input of one of the concatenated χ is (almost) fully non-zero, and the output of the same χ is fully zero. Those transitions are also possible in the case of Subterranean or Koala, however, the solver will never consider such solutions as the differential probability of such input difference would be 2^{-257} and therefore never lead to a trail with a low differential probability. In the case of ChiLow, such transitions are considered as the differential probability of such input would be close to $2^{\frac{n}{2}}$. Such transition only occurred when there are no two consecutive input bits with a zero difference and more specifically when the number of non-zero input differences is equal to $m-1$ where m is the size of χ. We experimentally verify that the following equation allows to remove all such transitions:

$$\sum_{i=0}^{m} y_i - \sum_{i=0}^{m} x_i \geq m+2$$

E.3 Differential Trails

In the following tables, we show examples of differential trails for one, two, and three rounds for ChiLow-$(32+\tau)$ and ChiLow-40 with differential probability following the result from 2. We show differences after each operation.

Table 8: 1-round differential trail of ChiLow-$(32+\tau)$.

	Data Path	Tweak Path
Δ_{in}	....	....1....
χ	....	....1....
L_{32}	....	...1.......1..1.
$\oplus$	...1.......1....................	

Table 9: 2-round differential trail of ChiLow-$(32+\tau)$.

	Data Path	Tweak Path
Δ_{in}	....	....1........................
χ	....	....1........................
L_{32}	....	..................................1....1.......1................
$\oplus$	....	
χ	....	..................................1....1.......1................
L_{32}	....	.......1...1....1.......1.........1....1.......1..........1....1
$\oplus$	.......1...1....1.......1.......	

Table 10: 3-round differential trail of ChiLow-$(32+\tau)$.

	Data Path	Tweak Path
Δ_{in}	....	1
χ	....	1
L_{32}	....	....1....1.......1........
$\oplus$	....	
χ	....	....1....1.....1.1........
L_{32}	....	.1.....1.1........1....1.......1...1....1.......1.........1.1..1
$\oplus$	.1.....1.1........1....1.......1	
χ	.1...1.1.1....1...1...11.....1.1	.1...111.1......111..1.1......1..111..1.1......11.......111.1.1.
L_{32}	1.11.......111.11.11...11.11.1.1	1..1.......11..11..1...11.11.1.111...1....1...1.11.......1111..1
$\oplus$	..1..........1....1.............	

Table 11: 1-round differential trail of ChiLow-40.

	Data Path	Tweak Path
Δ_{in}	....	....1..............
χ	....	....1..............
L_{32}	....	................1....1.......1..................................
$\oplus$	................1....1.......1..........	

Table 12: 2-round differential trail of ChiLow-40.

	Data Path	**Tweak Path**
Δ_{in}	....	...........1..
χ	....	...........1..
L_{32}	....	....1....1.......1....
$\oplus$	....	
χ	....	....1....1.......1....
L_{32}	....	...1.......1...1....1.......1.........1....1.......1..........1.
$\oplus$	...1.......1...1....1.......1.........1.	

Table 13: 3-round differential trail of ChiLow-40.

	Data Path	**Tweak Path**
Δ_{in}	....	.....1..
χ	....	.....1..
L_{32}	....	....1....1.......1......
$\oplus$	....	
χ	....	....1....1.......1......
L_{32}	....	......1.........1....1.......1..........1....1.......1...1....1.
$\oplus$	......1.........1....1.......1..........	
χ	......1.........1..1.1.......1..........	.....11.......111....1.........1........1....1.......1..11....1.
L_{32}	....11.11.1....1..1..1.1.......1..1.11..	.11.11...11.11.1............... 1.1.....11...11.1....11...1...1.1
$\oplus$	.11....111..11....1..1.1.........11.11.1	

F Details of Our MITM Attack

The procedure for mounting a MITM key-recovery attack on ChiLow-$(32 + \tau)$ is as follows:

1. Identify the bits fixed to constants and assign proper values to them.
2. Identify the key bits that are to be exhausted. Classify them into bits in blue and red.
3. In the computation of red bits, we assume that the blue bits are unknown and compute the internal state values based on the gray and red bits. This step is repeated for all the possible values of the red bits, and we store the corresponding computed information.
4. Compute blue bits in a similar manner.
5. Find matches between the store information obtained at Step 3 and Step 4. Check the matched keys using 4 pairs of plaintext-ciphertext.

For the 3-round attack, we capture this procedure in Algorithm 3.

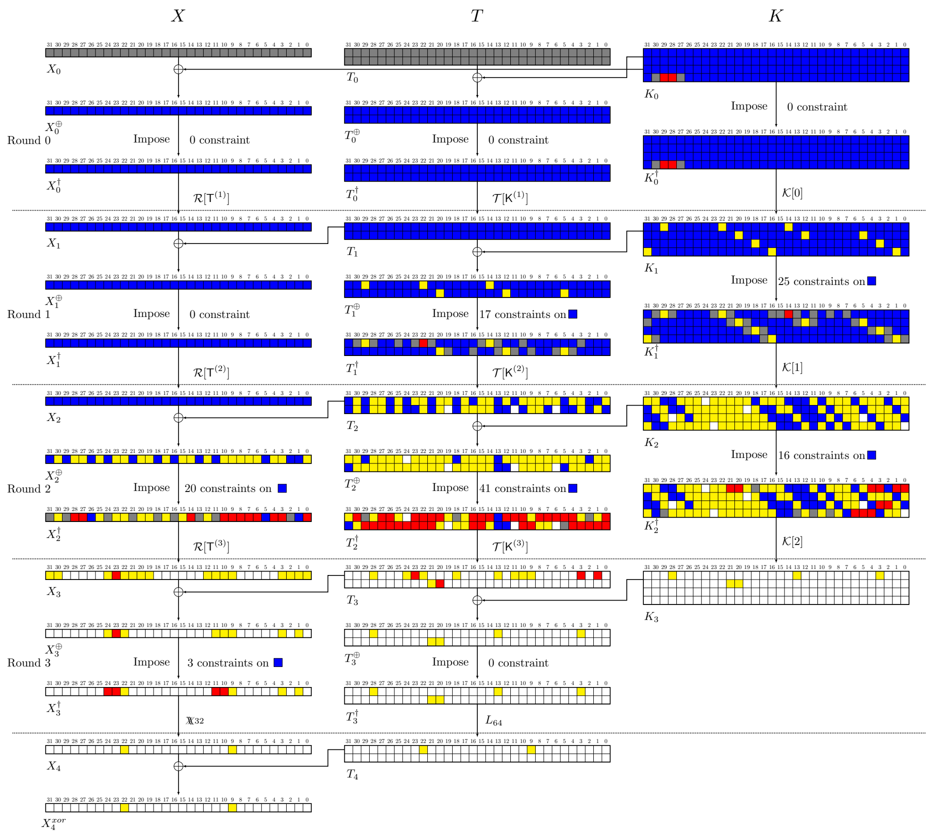


Fig. 6: MITM key-recovery attack on 4-round ChiLow-$(32 + \tau)$.

Algorithm 3: MITM key-recovery attack on 3-round ChiLow-(32+τ).

```
    Data: ${(M^i, T^i, C^i), i \in \{0,1,2,3\}}$
    Result: $K$
1   In Fig. 4, we have 29 non-linear equations on 🟥 in
      $K_1$:$\{f_0^{red}(K_1) = c_0^{red}, \ldots, f_{28}^{red}(K_1) = c_{28}^{red}\}$, 24 non-linear equations on 🟥 in
      $T_1^{\oplus}$:$\{f_{29}^{red}(T_1^{\oplus}) = c_{29}^{red}, \ldots, f_{52}^{red}(T_1^{\oplus}) = c_{52}^{red}\}$, and 17 non-linear equations on 🟥 in
      $X_2^{\oplus}$:$\{f_{53}^{red}(X_2^{\oplus}) = c_{53}^{red}, \ldots, f_{69}^{red}(X_2^{\oplus}) = c_{69}^{red}\}$;
2   In Fig. 4, we have 2 non-linear equations on 🟦 in
      $K_1$:$\{f_0^{blue}(K_1) = c_0^{blue}, f_1^{blue}(K_1) = c_1^{blue}\}$ and 6 non-linear equations on 🟦 in
      $T_1^{\oplus}$:$\{f_2^{blue}(T_1^{\oplus}) = c_2^{blue}, \ldots, f_7^{blue}(T_1^{\oplus}) = c_7^{blue}\}$;
3   Let $X_0 = M^0$ and $T_0 = T^0$;
4   for the 2 gray bits ⬛ $K_0[95,127] \in \mathbb{F}_2^2$ do
5       for the 95 red bits 🟥 $K_0[0-94] \in \mathbb{F}_2^{115}$ do
6           for the 8 constants $c_0^{blue}\| \ldots \|c_7^{blue} \in \mathbb{F}_2^8$ do
7               Use $K_0[95,127]$ to derive 🟥 and 🟨 in $K_1$;
8               Use $T$ and $K_1$ to derive 🟥 and 🟨 in $T_1^{\oplus}$;
9               We get 29 non-linear equations on $K_1$:$\{f_0(K_1) = c_0^{red}, \ldots, f_{28}(K_1) = c_{28}^{red}\}$;
10              We get 24 non-linear equations on
                  $T_1^{\oplus}$:$\{f_{29}(T_1^{\oplus}) = c_{29}^{red}, \ldots, f_{52}(T_1^{\oplus}) = c_{52}^{red}\}$;
11              Use $K_1^{\dagger}$ to derive 🟥 and 🟨 in $K_2$;
12              Use $T_1^{\dagger}$ and $K_2$ to derive 🟥 and 🟨 in $T_2^{\oplus}$;
13              Use $X$, $T$, $T_1$ and $T_2$ to derive 🟥 and 🟨 in $X_2^{\oplus}$;
14              We get 17 non-linear equations on
                  $X_2^{\oplus}$:$\{f_{53}(X_2^{\oplus}) = c_{53}^{red}, \ldots, f_{69}(X_2^{\oplus}) = c_{69}^{red}\}$;
15              Use $T_2^{\dagger}$ to derive 🟥 and 🟨 in $T_3$;
16              Use $X_2^{\dagger}$ and $T_3$ to derive 🟥 and 🟨 in $X_3^{\oplus}$;
17              Store the value of $K_0[0-94]$ in $L_{red}[X_3^{\oplus}[0-3, 6-9, 11-14, 16, 17, 19-$
                  $22, 24, 25, 29, 30]\|c_0^{red}\| \ldots \|c_{69}^{red}\|c_0^{blue}\| \ldots \|c_7^{blue}]$;

18      for the 31 blue bits 🟦 $K_0[96-126] \in \mathbb{F}_2^{31}$ do
19          for the 70 constants $c_0\| \ldots \|c_{69} \in \mathbb{F}_2^{69}$ do
20              Use $K_0[95,127]$ to derive 🟦 and 🟨 in $K_1$;
21              Use $T$ and $K_1$ to derive 🟨 in $T_1^{\oplus}$;
22              We get 2 non-linear equations on $K_1$:$\{f_0^{blue}(K_1) = c_0^{blue}, f_1^{blue}(K_1) = c_1^{blue}\}$;
23              We get 6 non-linear equations on
                  $T_1^{\oplus}$:$\{f_2^{blue}(T_1^{\oplus}) = c_2^{blue}, \ldots, f_7^{blue}(T_1^{\oplus}) = c_7^{blue}\}$;
24              Use $K_1^{\dagger}$ to derive 🟨 in $K_2$;
25              Use $T_1^{\dagger}$ and $K_2$ to derive 🟨 in $T_2^{\oplus}$;
26              Use $T_2$ to derive 🟨 in $X_2^{\dagger}$;
27              Use $X_2^{\dagger}$ and $T_3$ to derive 🟨 in $X_3^{\oplus}$;
28              for the $K_0[0-94]$ in
                  $L_{red}[(X_3^{\oplus}[0-3, 6-9, 11-14, 16, 17, 19-22, 24, 25, 29, 30] \oplus M^0[0-3, 6-$
                  $9, 11-14, 16, 17, 19-22, 24, 25, 29, 30])\|c_0^{red}\| \ldots \|c_{69}^{red}\|c_0^{blue}\| \ldots \|c_7^{blue}]$ do
29                  Use gray bits ⬛ $K_0[95,127]$, blue bits 🟦 $K_0[96-126]$, and red bits 🟥
                      $K_0[0-94]$ to derive a key candidate $K$;
30                  Use at most 4 plaintext-ciphertext pairs to check if there is a full match;
31                  if $D_K(T, C^0) \neq M^0$ then
32                      continue;
33                  if $D_K(T, C^1) \neq M^1$ then
34                      continue;
35                  if $D_K(T, C^2) \neq M^2$ then
36                      continue;
37                  if $D_K(T, C^3) = M^3$ then
38                      return $K$;
```

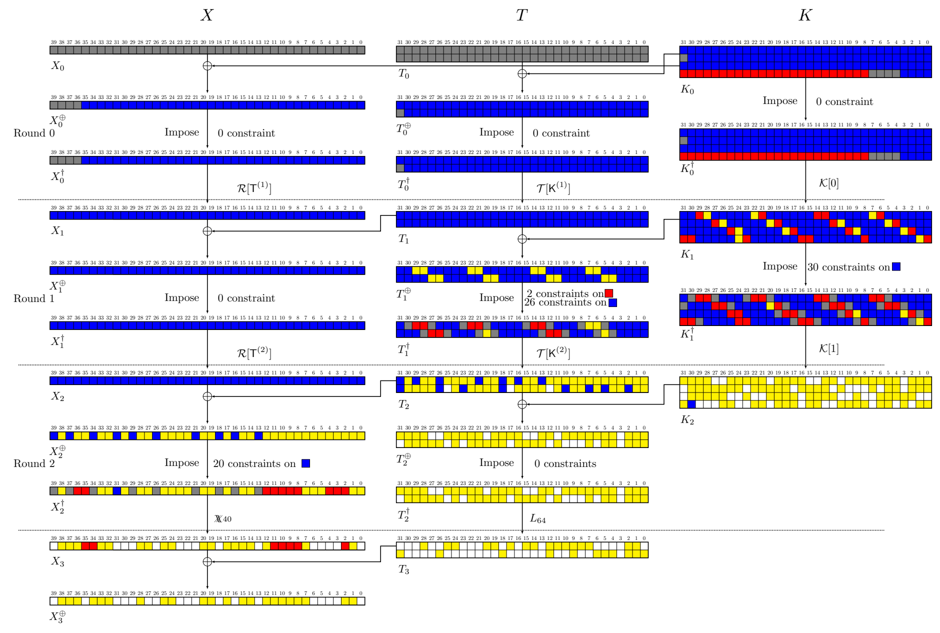


Fig. 7: MITM key-recovery attack on 3-round ChiLow-40.

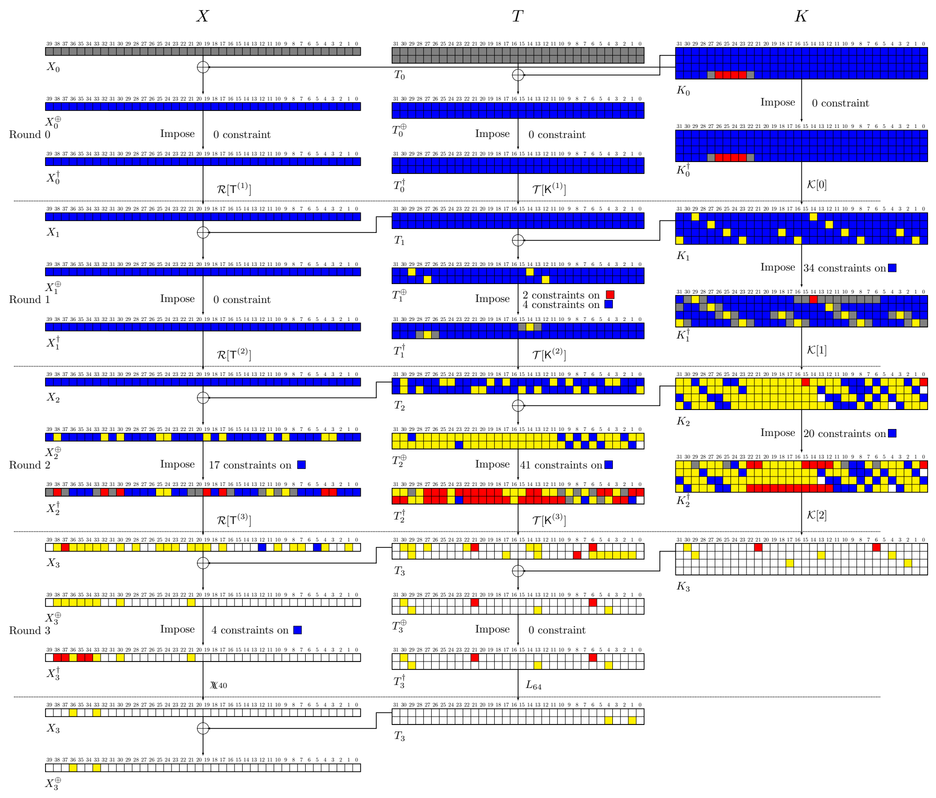


Fig. 8: MITM key-recovery attack on 4-round ChiLow-40.