Duel of the Titans: The Romulus and Remus Families of Lightweight AEAD Algorithms

Tetsu Iwata[1], Mustafa Khairallah[2], Kazuhiko Minematsu[4] and Thomas Peyrin[2,3]

[1] Nagoya University, Nagoya, Japan
tetsu.iwata@nagoya-u.jp

[2] Nanyang Technological University, Singapore, Singapore

[3] Temasek Laboratories@NTU, Singapore, Singapore
mustafam001@e.ntu.edu.sg,thomas.peyrin@ntu.edu.sg

[4] NEC, Kawasaki, Japan
k-minematsu@nec.com

Abstract. In this article, we propose two new families of very lightweight and efficient authenticated encryption with associated data (AEAD) modes, Romulus and Remus, that provide security beyond the birthday bound with respect to the block-length n. The former uses a tweakable block cipher (TBC) as internal primitive and can be proven secure in the standard model. The later uses a block cipher (BC) as internal primitive and can be proven secure in the ideal cipher model. Both our modes allow to switch very easily from the nonce-respecting to the nonce-misuse scenario.
Previous constructions, such as ΘCB3, are quite computationally efficient, yet needing a large memory for implementation, which makes them unsuitable for platforms where lightweight cryptography should play a key role. Romulus and Remus break this barrier by introducing a new architecture evolved from a BC mode COFB. They achieve the best of what can be possible with TBC – the optimal computational efficiency (rate-1 operation) and the minimum state size of a TBC mode (i.e., $(n + t)$-bit for n-bit block, t-bit tweak TBC), with almost equivalent provable security as ΘCB3. Actually, our comparisons show that both our designs present superior performances when compared to all other recent lightweight AEAD modes, being BC-based, TBC-based or sponge-based, in the nonce-respecting or nonce-misuse scenario.
We eventually describe how to instantiate Romulus and Remus modes using the Skinny lightweight tweakable block cipher proposed at CRYPTO 2016, including the hardware implementation results.

Keywords: Romulus and Remus · authenticated encryption · lightweight · tweakable block cipher.

1 Introduction

Lightweight cryptography has become a very active research domain, as the importance of pervasive computing and the Internet of Things (IoT) is growing. By lightweight, one usually refers to a primitive that allows compact implementations, i.e., minimizing the area required. In that direction, so-called serial implementations (implementations where the subcomponents of the round function are computed serially) will provide the smallest area on ASIC, at the cost of a much lower throughput. However, even though the area is certainly an important criterion for many applications, other performance measures should be taken into account, such as throughput, power and/or energy consumption, latency, etc. In particular, so-called round-based implementations (implementations where the entire round or more is computed in a cycle) are of paramount importance, as they are often the best implementation trade-off between power consumption and energy efficiency.

Many new ciphers, hash functions, and operating modes have been recently proposed with “lightweightness” as the main target. CAESAR competition [CAE] for authenticated encryption has received many submissions aiming at lightweightness, and two schemes (ACORN [Wu14] and Ascon [DEMS14]) were selected in the lightweight category. After several years of design/break process, NIST decided to organise a competition [NIS19] to identify the future lightweight authenticated encryption with associated data (AEAD) standard(s). One can separate the competition candidates into several classes: ad-hoc, sponge-based, block cipher-based, tweakable block cipher-based, etc.

Ad-hoc designs usually offer interesting performance features, but at the cost of lesser security guaranties on the general structure of the encryption. Lightweight sponge-based proposals have been flourishing, as they can offer sufficient security with a small internal state. However, as we will argue later in this article, their main drawback is a lower throughput as the internal permutation has to work on more than n bits, making them less energy/power efficient. Block cipher (BC)- based AE/AEAD designs have been studied for a long time, AES-GCM [MV04] and OCB [RBB03] being their most famous representing members. While they share the great advantage of being usable with widely deployed BC standards such as AES, most of them suffer from providing only birthday bound security, or low performance for beyond-birthday security. This is problematic for lightweight scenario, as 64-bit lightweight BCs seem hardly usable in order to provide sufficient security. Tweakable Block Ciphers (TBC) were introduced by Liskov et al. at CRYPTO 2002 [LRW02]. Since their inception, TBCs have been acknowledged as a powerful primitive as they can be used to construct simple yet highly secure Nonce-based or Misuse-Resistant Authenticated Encryption (NAE/MRAE) schemes, including ΘCB3 [KR11] and SCT [PS16]. Indeed, TBC-based AEAD schemes such as ΘCB3 are very efficient in terms of the number of primitive calls and are very secure as they achieve full n-bit security for block size of n bits (in contrary to most BC-based modes). In contrast, ΘCB3 needs an inverse of the TBC and a large state (the working memory beyond the primitive), which is not suitable for lightweight devices.

Our Contributions

In this work, we present two new TBC-based AEAD operating modes, named Romulus and Remus. They are lightweight, very efficient, and highly-secure NAE (Romulus-N and Remus-N) and MRAE (Romulus-M and Remus-M) schemes. The overall structure of Romulus and Remus shares similarity in part with a (TBC-based variant of) block cipher mode COFB [CIMN17a, CIMN17b], yet, we make numerous refinements to achieve our design goal.

The Romulus Mode. Our first proposal is Romulus. Its NAE version Romulus-N requires fewer TBC calls than ΘCB3 thanks to the faster MAC computation for associated data, while the hardware implementation is significantly smaller than ΘCB3 thanks to the reduced state size and inverse-freeness (i.e, TBC inverse is not needed). In fact, thanks to a careful integration of the TBC inside the mode, Romulus-N’s state size is comparable to what is needed for computing the TBC alone. Moreover, it encrypts an n-bit plaintext block by just one call of the n-bit block TBC, hence there is no efficiency loss. Romulus-N is extremely efficient for small messages, which is particularly important in many lightweight applications, requiring for example only 2 TBC calls to handle one associated data block and one message block (in comparison, other designs like ΘCB3, OCB3, TAE, CCM require from 3 to 5 TBC calls in the same situation).

Romulus achieves all these advantages without any security penalty, i.e., Romulus guarantees full n-bit security in the nonce-respecting model, which is a similar security bound to ΘCB3. In addition, the n-bit security of Romulus is proved under the standard model, which provides a high-level assurance for security not only quantitatively but also qualitatively. To elaborate a bit more, with a security proof in the standard model, one can precisely connect the security status of the primitive to the overall security of the mode that uses this primitive. In our case, for each of the members of Romulus, the best attack on it implies a chosen-plaintext attack (CPA) in the single-key setting against the underlying TBC, i.e., unless the TBC is broken by CPA adversaries in the single-key setting, Romulus indeed maintains the claimed n-bit security. Such a guarantee is not possible with non-standard models and it is often not easy to deduce the impact of a found "flaw" of the primitive to the security of the mode. In a more general context, this gap between the proof and the actual security is best exemplified by "uninstantiable" Random Oracle-Model schemes [CGH98]. To evaluate the security of Romulus, with the standard model proof, we can focus on the security evaluation of the TBC, while this type of focus is not possible in schemes with proofs in non-standard models. We stress that we are not discouraging the third party verification of the proofs, which is significantly important (see e.g. [IIMP19]).

If we compare Romulus-N's performance to other n-bit secure AE schemes that process n-bit of data, such as conventional sponge-based AEs using a $3n$-bit permutation with n-bit rate, the state size is comparable ($3n$ to $3.5n$ bits). Our advantage is that the underlying cryptographic primitive is expected to be much more lightweight and/or faster because of the smaller output size ($3n$ vs n bits). This efficiency is compared in Table 1. Similarly, Table 2 shows the efficiency of the misuse-resistant variant Romulus-M. Another interesting feature of Romulus is that it can reduce area depending on the use cases, without harming security. If it is enough to have a relatively short nonce or a short counter (or both), which is common to low-power networks, we can directly save the area by truncating the corresponding tweak length. This is possible if the internal TBC allows to reduce area if a part of its tweak is never used. A member of Romulus (Romulus-N2) particularly benefits from this feature. Note that this type of area reduction is not possible with conventional permutation/sponge-based AE schemes: it only offers a throughput/security trade-off. It goes without saying that these theoretical comparisons are meant to compare different modes without the effect of the underlying primitive and once the modes are instantiated and implemented, practical comparisons between different implementations are much more important (see Section 7).

The Remus Mode. Our second proposal is Remus. It shares its basic structure with Romulus and therefore also inherits its overall implementation advantages. The biggest difference of Remus from Romulus is the way it instantiates a TBC. Specifically, Remus takes an approach of utilizing the whole tweakey state [JNP14] of the TBC as a function of the key and tweak, using a tweak-dependent key derivation. In contrast to this, in Romulus, the TBC is used in the standard keying setting (i.e, tweakey state takes a persistent key material and a changing tweak). The tweak-dependent key derivation allows us to use a smaller variant of the TBC than those used by Romulus, and brings us better efficiency and comparable bit security. The downside is that the security proof of Remus is not based on the standard assumption of its cryptographic core (namely, the fact that TBC can be modelled as a tweakable pseudorandom permutation) as done for Romulus. Instead, we can prove the security of Remus by assuming the TBC as an ideal-cipher (thus ideal-cipher model proof), or, assuming the pseudorandomness of another TBC built on top of the standard BC, called ICE. The latter is a standard model proof but the assumption is still different from the pseudorandomness assumption of the TBC. This means that Remus is not a simple optimization of Romulus but is the consequence of trade-off between (qualitative) security and efficiency.

We specify a set of members for Remus that have different TBC (ICE) instantiations based on a block cipher in order to provide security-efficiency trade-offs. As with Romulus, the overall structure of Remus is partially similar to a TBC-variant of the block cipher mode COFB [CIMN17a, CIMN17b], yet, we make numerous refinements to achieve our design goal. Consequently, as a mode of TBC ICE, the NAE variant Remus-N achieves a significantly smaller state size than ΘCB3 [KR11], the typical choice for TBC-based AE mode, while keeping the equivalent efficiency (i.e., the same number of TBC calls). Also Remus is inverse-free (i.e, no TBC decryption routine is needed) unlike ΘCB3. For security, it allows either classical $n/2$-bit security or full n-bit security depending on the variant of ICE.

To see the superior performance of Remus-N, let us compare n-bit secure Remus-N2 with other n-bit secure AE schemes that process n bits of data, such as conventional sponge/permutation-based AEs using a $3n$-bit permutation with n-bit rate. Both have $3n$ state bits and process n-bit message block per primitive call. However, the cryptographic primitive for Remus-N2, which is an n-bit TBC with n-bit tweakey, is expected to be much more lightweight and/or faster because of smaller output size ($3n$ vs n bits). Since its tweakey is n bits, it is even smaller than the members of Romulus (they are n-bit secure and using tweakey state of $2n$ or $3n$ bits). Both sponge/permutation-based schemes and Remus rely on non-standard models (random permutation or ideal-cipher), and we emphasize that the security of the TBC instances that we propose to use inside Remus have been comprehensively evaluated, not only for the single-key related-tweak setting but also related-tweakey setting, which suggests strong reliability to be used as the ideal-cipher. Besides, we do not weaken the TBC instance from the original (say by reducing the number of rounds) in order to boost the throughput. This is a sharp difference from the strategy often taken in sponge/permutation-based constructions, where in order to boost the throughput the underlying permutation is made weaker than the stand-alone version for which the random permutation model is assumed.

An additional feature of Remus is that it offers a very flexible security/size trade-off without changing the throughput. In more detail, Remus contains $n/2$-bit secure variants (Remus-N1 and Remus-M1) and n-bit secure variants (Remus-N2 and Remus-M2). Their difference is only in the existence of the second (block) mask, which increases the state size. If the latter is too big and n-bit security is overkill, it is possible to derive an intermediate variant by truncating the second mask to (say) $n/2$ bits. It will be $(n + n/2)/2 = 3n/4$-bit secure. For simplicity, we did not include such variants in the official members of Remus, however, this flexibility would be useful in practice.

Misuse Resistance. Romulus-M and Remus-M are the MRAE versions of Romulus and Remus respectively, and they follow the general SIV construction [RS06]. However, they reuse the components of Romulus-N and Remus-N as much as possible, simply obtained by processing the message twice by Romulus-N or Remus-N. This allows a faster and smaller scheme than TBC-based MRAE SCT [PS16], yet, we maintain the strong security features of SCT. That is, Romulus-M achieves n-bit security against nonce-respecting adversaries and $n/2$-bit security against nonce-misusing adversaries, while a variant of Remus-M achieves the same security as the corresponding Remus-N variant against nonce-respecting

Table 1: Efficiency comparison of nonce-based AE schemes. λ is the bit security level of a mode. Here, (n, k)-BC is a block cipher of n-bit block and k-bit key, (n, t, k)-TBC is a TBC of n-bit block and k-bit key and t-bit tweak, and b-Perm is an b-bit cryptographic permutation.

Scheme	Number of Primitive Calls	Primitive	Security (λ)	State Size (S)	Rate (R)	Efficiency (S/R)	Inverse Free
Romulus-N1	$\lceil \frac{\lvert A\rvert-n}{2n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$	$(n, 1.5n, k)$-TBC[†], $n = k$	n	$n + 2.5k = 3.5\lambda$	1	3.5λ	Yes
Romulus-N2	$\lceil \frac{\lvert A\rvert-n}{1.75n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$	$(n, 1.2n, k)$-TBC[†], $n = k$	n	$n + 2.2k = 3.2\lambda$	1	3.2λ	Yes
Romulus-N3	$\lceil \frac{\lvert A\rvert-n}{1.75n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$	(n, n, k)-TBC, $n = k$	n	$n + 2k = 3\lambda$	1	3λ	Yes
Remus-N1	$\lceil \frac{\lvert A\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$	(n, k)-BC, $n = k$	$n/2$	$n + k = 4\lambda$[†]	1	4λ	Yes
Remus-N2	$\lceil \frac{\lvert A\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 2$	(n, k)-BC, $n = k$	n	$2n + k = 3\lambda$	1	3λ	Yes
Remus-N3	$\lceil \frac{\lvert A\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil$	(n, k)-BC, $n = k/2$	$n - 4$	$n + k = 3\lambda + 8$	1	$3\lambda + 8$	Yes
COFB [CIMN17a]	$\lceil \frac{\lvert A\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$	(n, k)-BC, $n = k$	$n/2 - \log_2 n/2$	$1.5n + k = 5.4\lambda$[‡]	1	5.4λ	Yes
ΘCB3 [♯] [KR11]	$\lceil \frac{\lvert A\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$	$(n, 1.5n, k)$-TBC, $n = k$	n	$2n + 2.5k = 4.5\lambda$	1	4.5λ	No
Beetle [CDNY18]	$\lceil \frac{\lvert A\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 2$	$2n$-Perm, $n = k$	$n - \log_2 n$	$2n = 2.12\lambda$	1/2	4.24λ	Yes
Ascon-128 [DEMS16]	$\lceil \frac{\lvert A\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$	$5n$-Perm, $n = k/2$	$n/2$	$7n = 3.5\lambda$	1/5	17.5λ	Yes
Ascon-128a [DEMS16]	$\lceil \frac{\lvert A\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$	$2.5n$-Perm, $n = k$	n	$3.5n = 3.5\lambda$	1/2.5	8.75λ	Yes
SpongeAE [♭] [BDPA11]	$\lceil \frac{\lvert A\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$	$3n$-Perm, $n = k$	n	$3n = 3\lambda$	1/3	9λ	Yes

[†] Can possibly be enhanced to 3λ with a different KDF and block cipher with $2k$-bit key;
[‡] Can possibly be enhanced to about 4λ with a $2n$-bit block cipher;
[♯] $1.5n$-bit tweak for n-bit nonce and $0.5n$-bit counter;
[♭] Duplex construction with n-bit rate, $2n$-bit capacity.

adversaries and $n/2$-bit security against nonce-misusing adversaries. Moreover, Romulus-M and Remus-M enjoy a very useful security feature called graceful degradation introduced in [PS16]. This ensures that the full security is almost retained if the number of nonce repetitions during encryption is limited (which is the main targeted scenario in practice). Thanks to the shared components, most of the advantages of Romulus-N and Remus-N mentioned above also hold for Romulus-M and Remus-M.

Instantiations with Skinny. As an underlying TBC, we adopt Skinny proposed at CRYPTO 2016 [BJK+16], leading to the two submissions Romulus [IKMP19b] and Remus [IKMP19a] for the NIST LWC standardization competition. See Appendix B for the details of the instantiation. We also instantiate Remus with the new TBC TGIF-BC, leading to another submission, TGIF [IKM+19]. The security of this TBC has been extensively studied, and it has attractive implementation characteristics.

2 Preliminaries

2.1 Notation

Let $\{0,1\}^*$ be the set of all finite bit strings, including the empty string ε. For $X \in \{0,1\}^*$, let $|X|$ denote its bit length. Here $|\varepsilon| = 0$. For an integer $n \geqslant 0$, let $\{0,1\}^n$ be the set of n-bit strings, and let $\{0,1\}^{\leqslant n} = \bigcup_{i=0,\ldots,n} \{0,1\}^i$, where $\{0,1\}^0 = \{\varepsilon\}$. Let $[\![n]\!] = \{1,\ldots,n\}$ and $[\![n]\!]_0 = \{0,1,\ldots,n-1\}$. Let $|X|_n = \max\{1, \lceil |X|/n \rceil\}$.

For two bit strings X and Y, $X \,\|\, Y$ is their concatenation. We also write this as XY if it is clear from the context. Let 0^i (1^i) be the string of i zero bits (i one bits), and for instance we write 10^i for $1 \,\|\, 0^i$. Bitwise XOR of two variables X and Y is denoted by $X \oplus Y$, where $|X| = |Y| = c$ for some positive integer c. For binary string X of $|X| \geqslant x$, we write $\texttt{lmt}_x(X)$ (resp. $\texttt{rmt}_x(X)$) to denote the leftmost (resp. rightmost) x bits of X.

Table 2: Efficiency comparison of misuse-resistant AE schemes. λ is the bit security level of a mode. Here, (n, k)-BC is a block cipher of n-bit block and k-bit key, (n, t, k)-TBC is a TBC of n-bit block and k-bit key and t-bit tweak. x, y in the security column means x-bit security in the nonce-respecting case, and y-bit security in the nonce-misuse case.

Scheme	**Number of Primitive Calls**	**Primitive**	**Security** (λ)	**State Size** (S)	**Rate** (R)	**Efficiency** (S/R)	**Inverse Free**
Romulus-M1	$\lceil \frac{\lvert A\rvert+\lvert M\rvert-n}{2n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$	$(n, 1.5n, k)$-TBC†, $n = k$	$n, n/2$	$n + 2.5k = 3.5\lambda$	2/3	5.25λ	Yes
Romulus-M2	$\lceil \frac{\lvert A\rvert+\lvert M\rvert-n}{1.75n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$	$(n, 1.2n, k)$-TBC†, $n = k$	$n, n/2$	$n + 2.2k = 3.2\lambda$	7/11	5λ	Yes
Romulus-M3	$\lceil \frac{\lvert A\rvert+\lvert M\rvert-n}{1.75n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$	(n, n, k)-TBC, $n = k$	$n, n/2$	$n + 2k = 3\lambda$	7/11	4.7λ	Yes
Remus-M1	$\lceil \frac{\lvert A\rvert+\lvert M\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$	(n, k)-BC, $n = k$	$n/2, n/2$	$2n = 4\lambda$	1/2	8λ	Yes
Remus-M2	$\lceil \frac{\lvert A\rvert+\lvert M\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 2$	(n, k)-BC, $n = k$	$n, n/2$	$3n = 3\lambda$	1/2	6λ	Yes
SCT † [PS16]	$\lceil \frac{\lvert A\rvert+\lvert M\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$	(n, n, k)-TBC, $n = k$	$n, n/2$	$4n = 4\lambda$	1/2	8λ	Yes
SUNDAE [BBLT18]	$\lceil \frac{\lvert A\rvert+\lvert M\rvert}{n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 1$	(n, k)-BC, $n = k$	$n/2, n/2$	$2n = 4\lambda$	1/2	8λ	Yes
ZAE $^\sharp$ [IMPS17]	$\lceil \frac{\lvert A\rvert+\lvert M\rvert}{2n} \rceil + \lceil \frac{\lvert M\rvert}{n} \rceil + 6$	(n, n, k)-TBC, $n = k$	n, n	$7n = 7\lambda$	1/2	14λ	Yes

† Tag is n bits;
$^\sharp$ Tag is $2n$ bits.

Note that we do not use "MSB" and "LSB" for them, which is customary but depends on endianness. Because our concrete specification (Appendix B) employs the little-endian format, our notation intends to avoid confusion. By convention, if one of X or Y is represented as an integer in $[\![2^c]\!]_0$ we assume a standard integer-to-binary encoding, i.e., an integer $\sum_{i=0}^{n-1} x_i 2^i$ for $x_i \in \{0,1\}$ is encoded to $(x_{n-1} \ldots x_1 x_0) \in \{0,1\}^n$. For example, $X \oplus 1$ denotes $X \oplus 0^{c-1}1$.

Padding. For $X \in \{0,1\}^{\leqslant l}$ of length multiple of 8 (i.e, byte string), let

$$\mathtt{pad}_l(X) = \begin{cases} X & \text{if } |X| = l, \\ X \,\|\, 0^{l-|X|-8} \,\|\, \mathtt{len}_8(X) & \text{if } 0 \leqslant |X| < l, \end{cases}$$

where $\mathtt{len}_8(X)$ denotes the one-byte encoding of the byte-length of X (we assume that $l < 256$ bytes). Here, $\mathtt{pad}_l(\varepsilon) = 0^l$. For example, when $l = 128$, $\mathtt{len}_8(X)$ has 16 variations (i.e., byte length 0 to 15), and we encode it to the last 4 bits of $\mathtt{len}_8(X)$ (for example, $\mathtt{len}_8(11) = \mathtt{00001011}$). The case $l = 64$ is similarly treated, by using the last 3 bits.

Throughout the paper, we assume inputs are byte strings, hence the above padding, which is efficient in hardware, works fine. At the end of Section 3.1, we comment how to extend the specifications to arbitrarily bit strings.

Parsing. For $X \in \{0,1\}^*$, let $(X[1], \ldots, X[x]) \xleftarrow{n} X$ be the parsing of X into n-bit blocks. Here, $X[1] \,\|\, X[2] \,\|\, \ldots \,\|\, X[x] = X$ and $x = |X|_n$. Note that $|X[x]| < n$ if $|X|$ is not a multiple of n. When $X = \varepsilon$, we have $X[1] \xleftarrow{n} X$ and $X[1] = \varepsilon$. Note in particular that $|\varepsilon|_n = 1$.

Alternating Parsing. Let n and t be positive integers larger than 8. For $X \in \{0,1\}^*$, let $(X[1], \ldots, X[x]) \xleftarrow{n,t} X$ be the parsing of X into n-bit blocks and t-bit blocks in an alternating order. That is, we have $X[1] \,\|\, X[2] \,\|\, \ldots \,\|\, X[x] = X$, where $|X[i]| = n$ for any odd $i \in \{1, \ldots, x-1\}$, $|X[i]| = t$ for any even $i \in \{1, \ldots, x-1\}$, $|X[x]| \in [\![n]\!]$ if x is odd, and $|X[x]| \in [\![t]\!]$ if x is even. When $X \neq \varepsilon$, x is determined as

$$x = \begin{cases} 2\lfloor |X|/(n+t) \rfloor & \text{if } |X| > 0 \text{ and } |X| \bmod (n+t) = 0 \\ 2\lfloor |X|/(n+t) \rfloor + 1 & \text{if } 1 \leqslant |X| \bmod (n+t) \leqslant n \\ 2\lfloor |X|/(n+t) \rfloor + 2 & \text{if } n < |X| \bmod (n+t) < n+t. \end{cases}$$

When $X = \varepsilon$, $X[1] \overset{n,t}{\leftarrow} X$ (thus $x = 1$) and $X[1] = \varepsilon$.

Galois Field. An element a in the Galois field $\mathrm{GF}(2^n)$ will be interchangeably represented as an n-bit string $a_{n-1} \dots a_1 a_0$, a formal polynomial $a_{n-1}\mathtt{x}^{n-1} + \cdots + a_1\mathtt{x} + a_0$, or an integer $\sum_{i=0}^{n-1} a_i 2^i$.

Matrix. Let G be an $n \times n$ binary matrix defined over GF(2). For $X \in \{0,1\}^n$, let $G(X)$ denote the matrix-vector multiplication over GF(2), where X is interpreted as a column vector. We may write $G \cdot X$ instead of $G(X)$.

(Tweakable) Block Cipher. A tweakable block cipher (TBC) is a keyed function $\widetilde{E} : \mathcal{K} \times \mathcal{T_W} \times \mathcal{M} \to \mathcal{M}$, where $\mathcal{K}$ is the key space, $\mathcal{T_W}$ is the tweak space, and $\mathcal{M} = \{0,1\}^n$ is the message space, such that for any $(K, T_w) \in \mathcal{K} \times \mathcal{T_W}$, $\widetilde{E}(K, T_w, \cdot)$ is a permutation over $\mathcal{M}$. We interchangeably write $\widetilde{E}(K, T_w, M)$ or $\widetilde{E}_K(T_w, M)$ or $\widetilde{E}_K^{T_w}(M)$. When $\mathcal{T_W}$ is singleton, it is essentially a block cipher and is simply written as $E : \mathcal{K} \times \mathcal{M} \to \mathcal{M}$.

2.2 Security Notions

Security Notions for NAE. We consider the standard security notions for nonce-based AE [BN08, BRW04, Rog04b]. Let Π denote an NAE scheme consisting of an encryption procedure $\Pi.\mathcal{E}_K$ and a decryption procedure $\Pi.\mathcal{D}_K$, for secret key K chosen uniform in the set $\mathcal{K}$ (denoted as $K \xleftarrow{\$} \mathcal{K}$). For plaintext M with nonce N and associated data A, $\Pi.\mathcal{E}_K$ takes (N, A, M) and returns ciphertext C (typically $|C| = |M|$) and tag T. For decryption, $\Pi.\mathcal{D}_K$ takes (N, A, C, T) and returns a decrypted plaintext M if the authentication check is successful, and otherwise an error symbol, $\perp$.

The privacy notion is the indistinguishability of encryption oracle $\Pi.\mathcal{E}_K$ from the random-bit oracle \$ which returns random $|M| + \tau$ bits for any query (N, A, M). The adversary is assumed to be nonce-respecting (NR), i.e., nonces can be arbitrarily chosen but must be distinct for encryption queries. We define the (NR) privacy advantage as

$$\mathbf{Adv}_{\Pi}^{\mathtt{priv}}(\mathcal{A}) \overset{\text{def}}{=} \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\Pi.\mathcal{E}_K(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot,\cdot)} \Rightarrow 1\right],$$

which measures the hardness of breaking the privacy notion for $\mathcal{A}$.

The authenticity notion is the probability of successful forgery via queries to $\Pi.\mathcal{E}_K$ and $\Pi.\mathcal{D}_K$ oracles. We define the (NR) authenticity advantage as

$$\mathbf{Adv}_{\Pi}^{\mathtt{auth}}(\mathcal{A}) \overset{\text{def}}{=} \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\Pi.\mathcal{E}_K(\cdot,\cdot,\cdot), \Pi.\mathcal{D}_K(\cdot,\cdot,\cdot,\cdot)} \text{ forges }\right],$$

where $\mathcal{A}$ forges if it receives a value $M' \neq \perp$ from $\Pi.\mathcal{D}_K$. Here, to prevent trivial wins, if $(C, T) \leftarrow \Pi.\mathcal{E}_K(N, A, M)$ is obtained earlier, $\mathcal{A}$ cannot query (N, A, C, T) to $\Pi.\mathcal{D}_K$. The adversary is assumed to be nonce-respecting for encryption queries.

Security Notions for MRAE. We adopt the security notions of MRAE following the same security definitions as above, with the exception that the adversary can now repeat nonces in encryption queries. Such an adversary is called nonce-misusing (NM)[1]. We write the NM-privacy advantage as

$$\mathbf{Adv}_{\Pi}^{\texttt{nm-priv}}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\Pi.\mathcal{E}_K(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot,\cdot)} \Rightarrow 1\right],$$

and the NM-authenticity advantage as

$$\mathbf{Adv}_{\Pi}^{\texttt{nm-auth}}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\Pi.\mathcal{E}_K(\cdot,\cdot,\cdot),\Pi.\mathcal{D}_K(\cdot,\cdot,\cdot,\cdot)} \text{ forges }\right].$$

We note that while NM adversaries can repeat nonces, without loss of generality, we assume that they do not repeat the same query. See also [RS06] for reference.

Security Notion for TBC. The security of TBC: $\mathcal{K} \times \mathcal{T} \times \mathcal{M} \to \mathcal{M}$ is defined by the indistinguishability from an ideal object, tweakable uniform random permutation (TURP), denoted by $\widetilde{\mathsf{P}}$, using chosen-plaintext, chosen-tweak queries. It is a set of independent uniform random permutations (URPs) over $\mathcal{M}$ indexed by tweak $T \in \mathcal{T}$. Let $\mathbf{Adv}_{\widetilde{E}}^{\texttt{tprp}}(\mathcal{A})$ denote the TPRP advantage of TBC $\widetilde{E}$ against adversary $\mathcal{A}$. It is defined as

$$\mathbf{Adv}_{\widetilde{E}}^{\texttt{tprp}}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\widetilde{E}_K(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\widetilde{\mathsf{P}}(\cdot,\cdot)} \Rightarrow 1\right].$$

An adversary against privacy (authenticity) notion is called privacy (authenticity) adversary, and if its attack resource is represented by the parameter list θ, we call it a θ-privacy (authenticity) adversary, assuming the semantics of θ is clearly understood. For example, θ may contain the number of queries and the total number of queried blocks etc. This holds for other security notions as well; e.g. we write (q_e, t)-TPRP-adversary to mean CPA adversary against TBC using q_e encryption queries and time complexity t.

3 Specifications

3.1 State Update Function

In this section, we specify our two new authenticated encryption modes, Romulus and Remus. First, we describe the state update function, that is common to both designs. Let G be an $n \times n$ binary matrix defined as an $n/8 \times n/8$ diagonal matrix of 8×8 binary sub-matrices G_s. They are defined as

$$G = \begin{pmatrix} G_s & 0 & 0 & \ldots & 0 \\ 0 & G_s & 0 & \ldots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & \ldots & 0 & G_s & 0 \\ 0 & \ldots & 0 & 0 & G_s \end{pmatrix}, \qquad G_s = \begin{pmatrix} 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \\ 1&0&0&0&0&0&0&1 \end{pmatrix},$$

where 0 for G represents the 8×8 zero matrix.

[1]It may also be called Nonce Repeating or Nonce Ignoring.

The state update function $\rho : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n \times \{0,1\}^n$ and its inverse $\rho^{-1} : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n \times \{0,1\}^n$ are defined as

$$\rho(S, M) = (S', C),$$

where $C = M \oplus G(S)$ and $S' = S \oplus M$. Similarly,

$$\rho^{-1}(S, C) = (S', M),$$

where $M = C \oplus G(S)$ and $S' = S \oplus M$. We note that we abuse the notation by writing ρ^{-1} as this function is only the inverse of ρ according to its second parameter. For any $(S, M) \in \{0,1\}^n \times \{0,1\}^n$, if $\rho(S, M) = (S', C)$ holds then $\rho^{-1}(S, C) = (S', M)$. Besides, we remark that $\rho(S, 0^n) = (S, G(S))$ holds.

Note that other state update functions are possible, but need to fulfill the following security condition. First, let $G^{(i)}$ be an $n \times n$ matrix that is equal to G except the $(i+1)$-st to n-th rows, which are set to all zero. Here, $G^{(0)}$ is the zero matrix and $G^{(n)} = G$, and for $X \in \{0,1\}^n$, $G^{(i)}(X) = \mathtt{lmt}_i(G(X)) \| 0^{n-i}$ for all $i = 0, 8, 16, \ldots, n$; note that all variables are byte strings, and $\mathtt{lmt}_i(X)$ is the leftmost $i/8$ bytes. Let I denote the $n \times n$ identity matrix.

Definition 1. We say that the $n \times n$ binary matrix G is *sound* if (1) G is regular (full-rank) and (2) $G^{(i)} + I$ is regular for all $i = 8, 16, \ldots, n$.

We have verified the soundness of our G proposal, for a range of n including $n = 64$ and $n = 128$, by a computer program.

Extension to Bit Strings. When the inputs are arbitrary bit strings rather than byte strings, we need $\mathtt{pad}_\ell(X)$ to be injective on all $X \in \{0,1\}^{<\ell}$ and keep $\mathtt{pad}_\ell(X) = X$ when $|X| = \ell$ as before. For example we can use the popular 10* padding. We also need to extend Definition 1 so that the second condition is changed to require $G^{(i)} + I$ is regular for all $i = 1, \ldots, n$. In fact, our G fulfills this for $n = 64, 128$.

3.2 Parameters

We use the following notation for the parameters of Remus and Romulus: nonce length nl, key length k, message block length n, AD block length $n + t$ for Romulus or n for Remus, counter bit length d, tag length τ. While we fix $\tau = n$, a tag for NAE schemes can be truncated if needed, at the cost of decreased security against forgery (see Section 4). We provide in Appendix B actual values of these parameters for our concrete proposed instances.

Remus uses as internal primitive a block cipher $E : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{M}$ with $\mathcal{M} = \{0,1\}^n$ and $\mathcal{K} = \{0,1\}^k$, and a mode $\mathtt{ICmode}$ to convert it into a TBC, $\mathtt{ICmode} \in \{\mathsf{ICE1}, \mathsf{ICE2}, \mathsf{ICE3}\}$ (for Ideal-Cipher Encryption). We use the term ICE to denote this family of TBCs. Each TBC is a mapping $: \mathcal{K} \times \mathcal{T} \times \mathcal{M} \rightarrow \mathcal{M}$, where $\mathcal{T} = \mathcal{N} \times \mathcal{D} \times \mathcal{B}$ is the tweak space with nonce space $\mathcal{N} = \{0,1\}^{nl}$, counter space $\mathcal{D} = [\![2^d - 1]\!]$, and $\mathcal{B} = [\![256]\!]_0$ for domain separation (i.e., deriving a small number of independent instances). Typically, $\mathcal{B}$ is represented as a byte.

On the other hand, Romulus uses a TBC $\widetilde{E} : \mathcal{K} \times \overline{\mathcal{T}} \times \mathcal{M} \rightarrow \mathcal{M}$, where $\mathcal{K} = \{0,1\}^k$, $\mathcal{M} = \{0,1\}^n$, and $\overline{\mathcal{T}} = \mathcal{T} \times \mathcal{B} \times \mathcal{D}$. Here, $\mathcal{T} = \{0,1\}^t$, $\mathcal{D} = [\![2^d - 1]\!]_0$, and $\mathcal{B} = [\![256]\!]_0$ for parameters t and d, and $\mathcal{B}$ is also represented as a byte. For tweak $\overline{T} = (T, B, D) \in \overline{\mathcal{T}}$, T is always assumed to be a byte string including ε, and t is a multiple of 8. $\mathcal{T}$ will be used to process the nonce or an AD block, $\mathcal{D}$ will be used for counter, while $\mathcal{B}$ is for domain separation. When the counter value is $i \in \mathcal{D}$, we write $\overline{i}$ to denote the i-th clocking of the counter as a part of the tweak (e.g. see Figure 2).

NAE and MRAE Families. Romulus and Remus each have two families, Romulus-N, Romulus-M, Remus-N and Remus-M, and each family consists of several members (the sets of parameters). The x-N families implement nonce-based AE (NAE) secure against Nonce-respecting adversaries, and the x-M families implement nonce Misuse-resistant AE (MRAE) introduced by Rogaway and Shrimpton [RS06]. The names Romulus and Remus stand for the corresponding set of two families.

3.3 TBC ICE for Remus

As described above, Remus uses ICE which consists of three variants: ICE1, ICE2 and ICE3. Each variant consists of two main components, the key derivation function $\mathsf{KDF} : \mathcal{K} \times \mathcal{N} \to \mathcal{L} \times \mathcal{V}$, and the "core" encryption function $\mathsf{ICEnc} : (\mathcal{L} \times \mathcal{V}) \times (\mathcal{D} \times \mathcal{B}) \times \mathcal{M} \to \mathcal{M}$. Here, $\mathcal{L} = \mathcal{K} = \{0,1\}^k$ and $\mathcal{V} = \mathcal{M} = \{0,1\}^n$. The algorithm of ICEnc is shown in Figure 1 for all variants (and a more graphical representation is given in Figures 7, 8 and 9). In addition, there is a tweakey encoding function $\mathtt{encode} : \mathcal{L} \times \mathcal{D} \times \mathcal{B} \to \mathcal{K}$ inside ICEnc. For convenience, KDF for ICE1 may also be referred to as KDF1 (KDF2 and KDF3 are defined analogously).

An encryption with ICE is performed as follows. Given a tweak $T = (N, D, B) \in \mathcal{T}$, a key $K \in \mathcal{K}$, and a plaintext $M \in \mathcal{M}$, we first derive the nonce-dependent mask values (L, V) with $\mathsf{KDF}(N, K) \to (L, V)$, and then ICEnc encrypts M as $\mathsf{ICEnc}(L, V, D, B, M) \to C$, using the key of the internal E determined by $\mathtt{encode}(L, D, B)$. Here, $\mathsf{ICEnc}(L, V, D, B, *)$ is a permutation over $\mathcal{M}$ for any (L, V, D, B). Each variant is then defined as follows:

1. ICE1: $n = d = k$, $nl \leqslant n$.
   (a) $\mathsf{KDF}(N, K) = (L, V)$ where $L = G(E_K(N \,\|\, 0^{n-nl}))$, $V = 0^n$.
   (b) $\mathtt{encode}(L, D, B) = 2^D L \oplus B$, where multiplication is over GF(2^n).
2. ICE2: $n = d = k$, $nl \leqslant n$.
   (a) $\mathsf{KDF}(N, K) = (L, V)$ where $L' = E_K(N \,\|\, 0^{n-nl}), V' = E_{K \oplus 1}(L')$, and $L = G(L')$ and $V = G(V')$.
   (b) $\mathtt{encode}(L, D, B) = 2^D L \oplus B$, where multiplication is over GF(2^n).
3. ICE3: $nl \leqslant k - 8$, $d = k - 8$, n is arbitrary (however, it matters for the security of the whole Remus).
   (a) $\mathsf{KDF}(N, K) = (L, V)$ where $L \leftarrow (N \,\|\, 0^{k-nl}) \oplus K, V \leftarrow 0^n$.
   (b) $\mathtt{encode}(L, D, B) = (2^D L[1]) \,\|\, (B \oplus L[2])$, where $(L[1], L[2]) \overset{k-8}{\leftarrow} L$ and the multiplication is over GF(2^{k-8}) and applied to $L[1]$. Note that $|L[1]| = k - 8$ and $|L[2]| = 8$.

Note that ICE1 and ICE2 differ only in the second mask V derived by their KDFs.

When ICE is working inside Remus, the corresponding KDF is performed only once as an initialization. For ICE1 or ICE2, KDF involves one or two calls of E and matrix multiplications by G (see above). For ICE3, KDF is just a linear operation of (N, K). For each input block, ICE applies doubling to the derived mask values. Since doubling is a sequential operation, computing $\mathsf{ICEnc}_{L,V}^{D+1,B}(M)$ after $\mathsf{ICEnc}_{L,V}^{D,B'}(M')$ is easy and does not need any additional memory.

```
Algorithm $\mathsf{ICEnc}_{L,V}^{D,B}(M)$
  1. $S \leftarrow 2^D V \oplus M$
  2. $T_K \leftarrow \mathtt{encode}(L, D, B)$
  3. $S \leftarrow E_{T_K}(S)$
  4. $C \leftarrow 2^D V \oplus S$
  5. return $C$
```

Figure 1: Definition of ICEnc, the core encryption routine of ICE. ICEnc is common to all three variants of ICE, ICE1 and ICE2 and ICE3 except the definition of `encode`. Note that ICE1 and ICE3 fix $V = 0^n$, hence effectively $S \leftarrow M$ (line 1) and $C \leftarrow S$ (line 4). Variables L and V are assumed to be derived from the corresponding KDF taking (N, K), as a pre-processing. KDFs are defined in Section 3.3.

3.4 Romulus

The specification of the NAE mode Romulus-N is shown in Figure 2, while Figure 3 gives a more graphical representation. Similarly, the specification of the MRAE mode Romulus-M is shown in Figure 4, while Figure 5 gives a more graphical representation.

To encrypt (N, A, M) under key K, in Romulus-N, we first hash $A = (A[1], \dots, A[a])$ into the state S in line 11 in Figure 2, where $A[1], A[3], A[5], \dots$ are injected into the state with ρ, and $A[2], A[4], A[6], \dots$ are processed with the TBC. We then use nonce N to compute $S = \widetilde{E}_K^{(N, w_A, \overline{a})}(S)$, that could be seen as the nonce-dependent MAC value of A. Then M is processed with ρ to generate C, where we keep using the TBC that takes N as a part of the input. The tag T is generated as $T = G(\widetilde{E}_K^{(N, w_M, \overline{m})}(S))$ from the final state S after the process of M.

In Romulus-M, we first hash both A and M into the state S in line 20 or 22 in Figure 4, and then the tag T is computed as $T = G(\widetilde{E}_K^{(N, w, \overline{a+m})}(S))$ in line 24. The tag generation may seem complex in appearance, but it simply follows the same process as Romulus-N to hash both A and M, and the encryption part of M is similar to Romulus-N. Note that the algorithm always assumes $t = nl$. These figures adopt the concrete values of domain separation (in the second argument of the tweak) described in Appendix B.

Although the pseudocode is identical, each member of Romulus-N and Romulus-M shown in Tables 1 and 2 has a distinct tweakey encoding, and the underlying TBCs are not identical in its tweak length. Each member is designed to achieve a unique tread-off between security, efficiency, and usability (mainly on the support of maximum input length). See Appendix B for the concrete specifications of the members of Romulus.

3.5 Remus

The specification of the nonce-based AE mode Remus-N is shown in Figure 6, and a more graphical representation is given in Figures 7, 8, 9. Similarly, the specification of the misuse-resistant AE Remus-M is shown in Figure 10, while a more graphical representation is given in Figures 11 and 12.

The basic procedure of Remus-N and Remus-M follows that of Romulus-N and Romulus-M, respectively, while we use ICE as the underlying primitive that does not take A nor M as the input. As with Romulus, the figures adopt the concrete domain separation values from Appendix B and use $\overline{i}$ to denote the counter value i.

```
Algorithm Romulus-N.Enc_K(N, A, M)
 1. $S \leftarrow 0^n$
 2. $(A[1], \ldots, A[a]) \xleftarrow{n,t} A$
 3. if $a \bmod 2 = 0$ then $u \leftarrow t$ else $n$
 4. if $|A[a]| < u$ then $w_A \leftarrow 26$ else $24$
 5. $A[a] \leftarrow \mathtt{pad}_u(A[a])$
 6. for $i = 1$ to $\lfloor a/2 \rfloor$
 7.     $(S, \eta) \leftarrow \rho(S, A[2i-1])$
 8.     $S \leftarrow \widetilde{E}_K^{(A[2i],8,\overline{2i-1})}(S)$
 9. end for
10. if $a \bmod 2 = 0$ then $V \leftarrow 0^n$ else $A[a]$
11. $(S, \eta) \leftarrow \rho(S, V)$
12. $S \leftarrow \widetilde{E}_K^{(N,w_A,\overline{a})}(S)$
13. $(M[1], \ldots, M[m]) \xleftarrow{n} M$
14. if $|M[m]| < n$ then $w_M \leftarrow 21$ else $20$
15. for $i = 1$ to $m - 1$
16.     $(S, C[i]) \leftarrow \rho(S, M[i])$
17.     $S \leftarrow \widetilde{E}_K^{(N,4,\overline{i})}(S)$
18. end for
19. $M'[m] \leftarrow \mathtt{pad}_n(M[m])$
20. $(S, C'[m]) \leftarrow \rho(S, M'[m])$
21. $C[m] \leftarrow \mathtt{lmt}_{|M[m]|}(C'[m])$
22. $S \leftarrow \widetilde{E}_K^{(N,w_M,\overline{m})}(S)$
23. $(\eta, T) \leftarrow \rho(S, 0^n)$
24. $C \leftarrow C[1] \,\|\, \ldots \,\|\, C[m-1] \,\|\, C[m]$
25. return $(C, T)$
```

```
Algorithm Romulus-N.Dec_K(N, A, C, T)
 1. $S \leftarrow 0^n$
 2. $(A[1], \ldots, A[a]) \xleftarrow{n,t} A$
 3. if $a \bmod 2 = 0$ then $u \leftarrow t$ else $n$
 4. if $|A[a]| < u$ then $w_A \leftarrow 26$ else $24$
 5. $A[a] \leftarrow \mathtt{pad}_u(A[a])$
 6. for $i = 1$ to $\lfloor a/2 \rfloor$
 7.     $(S, \eta) \leftarrow \rho(S, A[2i-1])$
 8.     $S \leftarrow \widetilde{E}_K^{(A[2i],8,\overline{2i-1})}(S)$
 9. end for
10. if $a \bmod 2 = 0$ then $V \leftarrow 0^n$ else $A[a]$
11. $(S, \eta) \leftarrow \rho(S, V)$
12. $S \leftarrow \widetilde{E}_K^{(N,w_A,\overline{a})}(S)$
13. $(C[1], \ldots, C[m]) \xleftarrow{n} C$
14. if $|C[m]| < n$ then $w_C \leftarrow 21$ else $20$
15. for $i = 1$ to $m - 1$
16.     $(S, M[i]) \leftarrow \rho^{-1}(S, C[i])$
17.     $S \leftarrow \widetilde{E}_K^{(N,4,\overline{i})}(S)$
18. end for
19. $\widetilde{S} \leftarrow (0^{|C[m]|} \,\|\, \mathtt{rmt}_{n-|C[m]|}(G(S)))$
20. $C'[m] \leftarrow \mathtt{pad}_n(C[m]) \oplus \widetilde{S}$
21. $(S, M'[m]) \leftarrow \rho^{-1}(S, C'[m])$
22. $M[m] \leftarrow \mathtt{lmt}_{|C[m]|}(M'[m])$
23. $S \leftarrow \widetilde{E}_K^{(N,w_C,\overline{m})}(S)$
24. $(\eta, T^*) \leftarrow \rho(S, 0^n)$
25. $M \leftarrow M[1] \,\|\, \ldots \,\|\, M[m-1] \,\|\, M[m]$
26. if $T^* = T$ then return $M$ else $\perp$
```

```
Algorithm $\rho(S, M)$
 1. $C \leftarrow M \oplus G(S)$
 2. $S' \leftarrow S \oplus M$
 3. return $(S', C)$
```

```
Algorithm $\rho^{-1}(S, C)$
 1. $M \leftarrow C \oplus G(S)$
 2. $S' \leftarrow S \oplus M$
 3. return $(S', M)$
```

Figure 2: The Romulus-N nonce-based AE mode. Lines of [**if (statement) then** $X \leftarrow x$ **else** x'] are shorthand for [**if (statement) then** $X \leftarrow x$ **else** $X \leftarrow x'$]. The dummy variable η is always discarded. We use Romulus-N1 as working example. For other Romulus-N members, the values of the bits b_7 and b_6 in the domain separation (specified in Appendix B) need to be adapted accordingly.

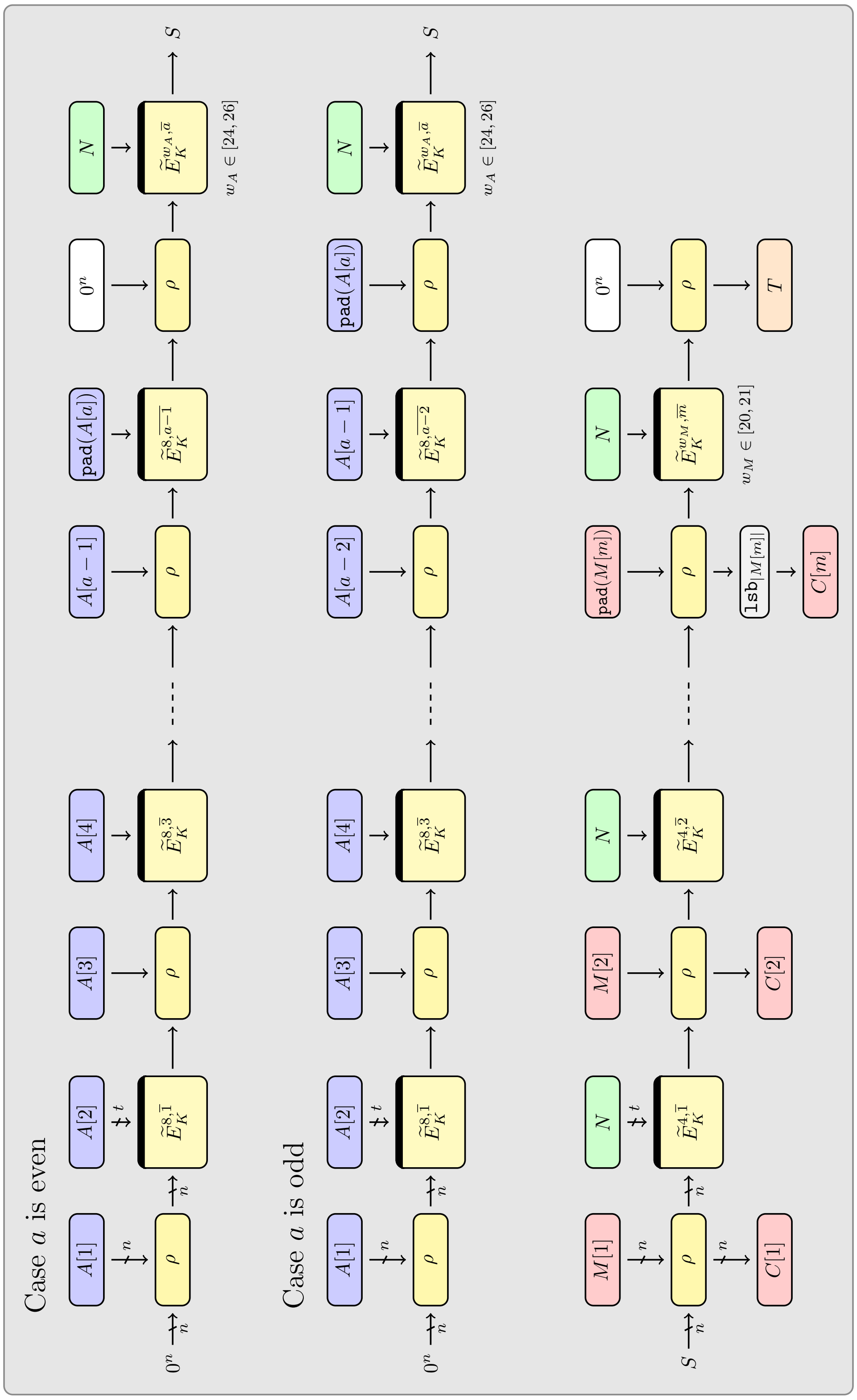
4 Security Analysis

In this section, we provide security bounds of Romulus and Remus. All the corresponding proofs will be given in Appendix A. We consider NAE notions for Romulus-N, and both NAE and MRAE notions for Romulus-M, that is, both nonce-respecting (NR) and nonce-misusing (NM) adversaries.

4.1 Security of Romulus

4.1.1 Security of Romulus-N

For $A \in \{0,1\}^*$, we say A has a AD blocks if it is parsed as $(A[1], \ldots, A[a]) \xleftarrow{n,t} A$. Let $\tilde{a} = \lfloor a/2 \rfloor + 1$ which is a bound of the actual number of primitive calls for AD. Similarly, for plaintext $M \in \{0,1\}^*$, we say M has m message blocks if $|M|_n = m$. The same applies to ciphertext C. For encryption query (N, A, M) or decryption query (N, A, C, T) of a AD blocks and m message blocks, the number of total TBC calls is at most $\tilde{a} + m$, which is called the number of *effective blocks* of a query.



Case a is even
0^n
n
$A[1]$
ρ
$A[2]$
t
$\widetilde{E}_K^{8,\overline{1}}$
$A[3]$
$A[4]$
$\widetilde{E}_K^{8,\overline{3}}$
$A[a-1]$
$\mathtt{pad}(A[a])$
$\widetilde{E}_K^{8,\overline{a-1}}$
0^n
N
$\widetilde{E}_K^{w_A,\overline{a}}$
$w_A \in [24, 26]$
S
Case a is odd
0^n
$A[1]$
$A[2]$
$\widetilde{E}_K^{8,\overline{1}}$
$A[3]$
$A[4]$
$\widetilde{E}_K^{8,\overline{3}}$
$A[a-2]$
$A[a-1]$
$\widetilde{E}_K^{8,\overline{a-2}}$
$\mathtt{pad}(A[a])$
N
$\widetilde{E}_K^{w_A,\overline{a}}$
$w_A \in [24, 26]$
S
S
$M[1]$
$C[1]$
N
$\widetilde{E}_K^{4,\overline{1}}$
$M[2]$
$C[2]$
N
$\widetilde{E}_K^{4,\overline{2}}$
$\mathtt{pad}(M[m])$
$\mathtt{lsb}_{|M[m]|}$
$C[m]$
N
$\widetilde{E}_K^{w_M,\overline{m}}$
$w_M \in [20, 21]$
0^n
T

Figure 3: The Romulus-N nonce-based AE mode. (Top) process of AD with an even number of AD blocks. (Middle) process of AD with an odd number of AD blocks. (Bottom) Encryption. We use Romulus-N1 as working example. For other Romulus-N members, the values of the bits b_7 and b_6 in the domain separation need to be adapted accordingly.

```
Algorithm Romulus-M.Enc_K(N, A, M)
 1. $S \leftarrow 0^n$
 2. $(X[1], \dots, X[a]) \overset{n,t}{\leftarrow} A$
 3. if $a \bmod 2 = 0$ then $u \leftarrow t$ else $n$
 4. $(X[a+1], \dots, X[a+m]) \overset{n+t-u,u}{\longleftarrow} M$
 5. if $m \bmod 2 = 0$ then $v \leftarrow u$ else $n+t-u$
 6. $w \leftarrow 48$
 7. if $|X[a]| < u$ then $w \leftarrow w \oplus 2$
 8. if $|X[a+m]| < v$ then $w \leftarrow w \oplus 1$
 9. if $a \bmod 2 = 0$ then $w \leftarrow w \oplus 8$
10. if $m \bmod 2 = 0$ then $w \leftarrow w \oplus 4$
11. $X[a] \leftarrow \mathtt{pad}_u(X[a])$
12. $X[a+m] \leftarrow \mathtt{pad}_v(X[a+m])$
13. $x \leftarrow 40$
14. for $i = 1$ to $\lfloor (a+m)/2 \rfloor$
15.     $(S, \eta) \leftarrow \rho(S, X[2i-1])$
16.     if $i = \lfloor a/2 \rfloor + 1$ then $x \leftarrow x \oplus 4$
17.     $S \leftarrow \widetilde{E}_K^{(X[2i],x,\overline{2i-1})}(S)$
18. end for
19. if $a \bmod 2 = m \bmod 2$ then
20.     $(S, \eta) \leftarrow \rho(S, 0^n)$
21. else
22.     $(S, \eta) \leftarrow \rho(S, X[a+m])$
23. $S \leftarrow \widetilde{E}_K^{(N,w,\overline{a+m})}(S)$
24. $(\eta, T) \leftarrow \rho(S, 0^n)$
25. if $M = \epsilon$ then return $(\epsilon, T)$
26. $S \leftarrow T$
27. $(M[1], \dots, M[m']) \overset{n}{\leftarrow} M$
28. $z \leftarrow |M[m']|$
29. $M[m'] \leftarrow \mathtt{pad}_n(M[m'])$
30. for $i = 1$ to $m'$
31.     $S \leftarrow \widetilde{E}_K^{(N,36,\overline{i-1})}(S)$
32.     $(S, C[i]) \leftarrow \rho(S, M[i])$
33. end for
34. $C[m'] \leftarrow \mathtt{lsb}_z(C[m'])$
35. $C \leftarrow C[1] \,\|\, \dots \,\|\, C[m'-1] \,\|\, C[m']$
36. return $(C, T)$
```

```
Algorithm Romulus-M.Dec_K(N, A, C, T)
 1. if $C = \epsilon$ then $M \leftarrow \epsilon$
 2. else
 3.     $S \leftarrow T$
 4.     $(C[1], \dots, C[m']) \overset{n}{\leftarrow} C$
 5.     $z \leftarrow |C[m']|$
 6.     $C[m'] \leftarrow \mathtt{pad}_n(C[m'])$
 7.     for $i = 1$ to $m'$
 8.         $S \leftarrow \widetilde{E}_K^{(N,36,\overline{i-1})}(S)$
 9.         $(S, M[i]) \leftarrow \rho^{-1}(S, C[i])$
10.     end for
11.     $M[m'] \leftarrow \mathtt{lsb}_z(M[m'])$
12.     $M \leftarrow M[1] \,\|\, \dots \,\|\, M[m'-1] \,\|\, M[m']$
13. $S \leftarrow 0^n$
14. $(X[1], \dots, X[a]) \overset{n,t}{\leftarrow} A$
15. if $a \bmod 2 = 0$ then $u \leftarrow t$ else $n$
16. $(X[a+1], \dots, X[a+m]) \overset{n+t-u,u}{\longleftarrow} M$
17. if $m \bmod 2 = 0$ then $v \leftarrow u$ else $n+t-u$
18. $w \leftarrow 48$
19. if $|X[a]| < u$ then $w \leftarrow w \oplus 2$
20. if $|X[a+m]| < v$ then $w \leftarrow w \oplus 1$
21. if $a \bmod 2 = 0$ then $w \leftarrow w \oplus 8$
22. if $m \bmod 2 = 0$ then $w \leftarrow w \oplus 4$
23. $X[a] \leftarrow \mathtt{pad}_u(X[a])$
24. $X[a+m] \leftarrow \mathtt{pad}_v(X[a+m])$
25. $x \leftarrow 40$
26. for $i = 1$ to $\lfloor (a+m)/2 \rfloor$
27.     $(S, \eta) \leftarrow \rho(S, X[2i-1])$
28.     if $i = \lfloor a/2 \rfloor + 1$ then $x \leftarrow x \oplus 4$
29.     $S \leftarrow \widetilde{E}_K^{(X[2i],x,\overline{2i-1})}(S)$
30. end for
31. if $a \bmod 2 = m \bmod 2$ then
32.     $(S, \eta) \leftarrow \rho(S, 0^n)$
33. else
34.     $(S, \eta) \leftarrow \rho(S, X[a+m])$
35. $S \leftarrow \widetilde{E}_K^{(N,w,\overline{a+m})}(S)$
36. $(\eta, T^*) \leftarrow \rho(S, 0^n)$
37. if $T^* = T$ then return $M$ else $\perp$
```

```
Algorithm $\rho(S, M)$
 1. $C \leftarrow M \oplus G(S)$
 2. $S' \leftarrow S \oplus M$
 3. return $(S', C)$
```

```
Algorithm $\rho^{-1}(S, C)$
 1. $M \leftarrow C \oplus G(S)$
 2. $S' \leftarrow S \oplus M$
 3. return $(S', M)$
```

Figure 4: The Romulus-M misuse-resistant AE mode. Lines of [**if (statement) then** $X \leftarrow x$ **else** x'] are shorthand for [**if (statement) then** $X \leftarrow x$ **else** $X \leftarrow x'$]. The dummy variable η is always discarded. We use Romulus-M1 as working example. For other Romulus-M members, the values of the bits b_7 and b_6 in the domain separation (specified in Appendix B) need to be adapted accordingly. Note that in the case of empty message, no encryption call has to be performed in the encryption part.

While the specification assumes an n-bit tag, we extend it to be (arbitrarily) fixed truncated to $\tau \in [\![n]\!]$ bits, and show the bounds for the case of a τ-bit tag.

Theorem 1. *Let $\mathcal{A}$ be an NR privacy adversary against* Romulus-N *with time complexity t_A and with total number of effective blocks σ_{priv}. Moreover, let $\mathcal{B}$ be an NR authenticity adversary using q_d decryption queries, with total number of effective blocks for encryption*

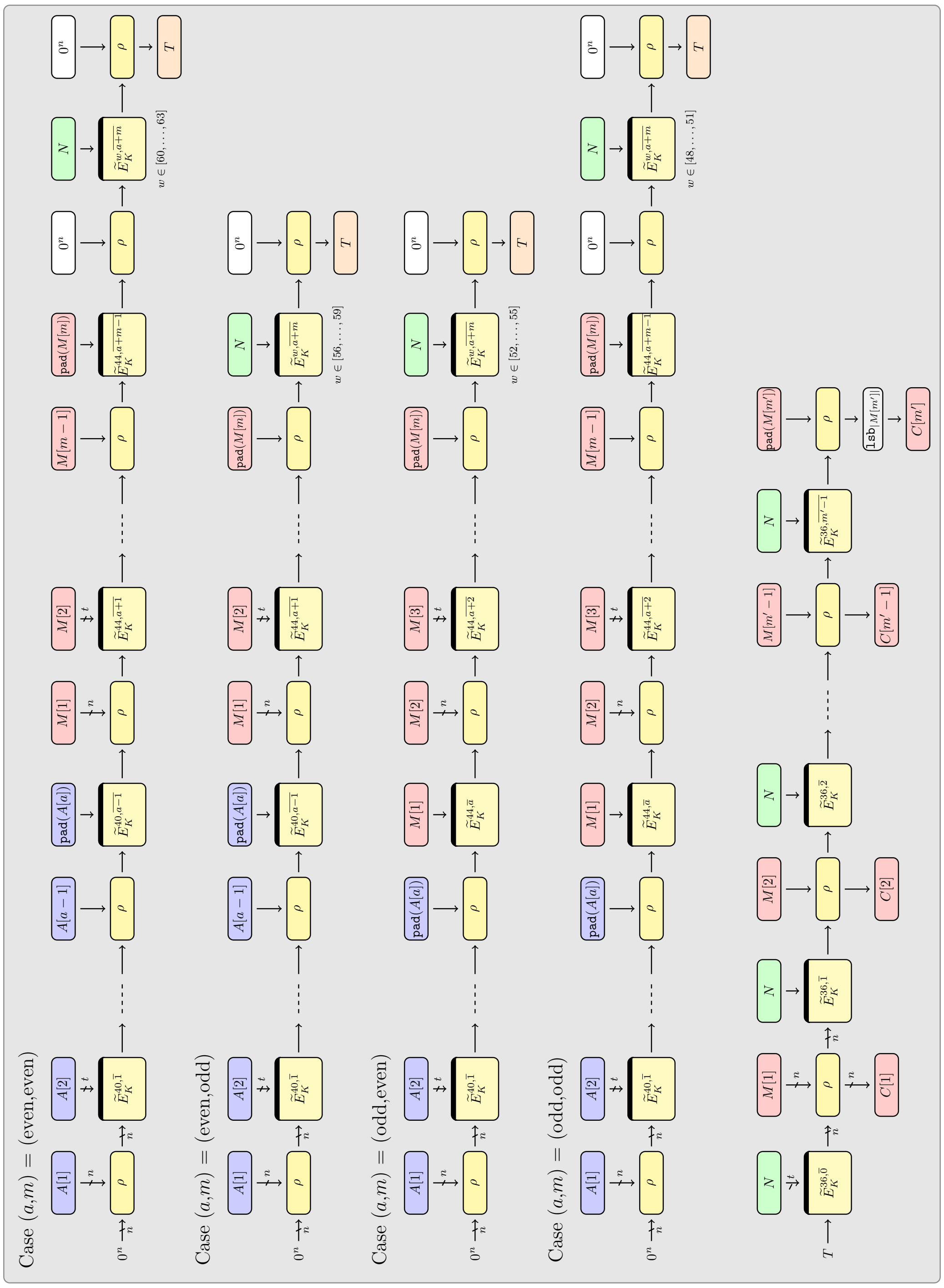


Figure 5: The Romulus-M misuse-resistant AE mode. (Top) process of AD with an even/even, even/odd, odd/even, odd/odd number of AD blocks and M blocks, respectively. (Bottom) Encryption. We use Romulus-M1 as working example. For other Romulus-M members, the values of the bits b_7 and b_6 in the domain separation need to be adapted accordingly.

```
Algorithm $\mathsf{Remus\text{-}N.Enc}_K(N, A, M)$
 1. $(L, V) \leftarrow \mathsf{KDF}(N, K)$
 2. $S \leftarrow 0^n$
 3. $(A[1], \ldots, A[a]) \xleftarrow{n} A$
 4. $(M[1], \ldots, M[m]) \xleftarrow{n} M$
 5.  if $|A[a]| < n$ then $w_A \leftarrow 13$ else $12$
 6.  if $|M[m]| < n$ then $w_M \leftarrow 11$ else $10$
 7. $A[a] \leftarrow \mathtt{pad}_n(A[a])$
 8. for $i = 1$ to $a - 1$
 9.     $(S, \eta) \leftarrow \rho(S, A[i])$
10.     $S \leftarrow \mathsf{ICEnc}^{\overline{i},4}_{L,V}(S)$
11. end for
12. $(S, \eta) \leftarrow \rho(S, A[a])$
13. $S \leftarrow \mathsf{ICEnc}^{\overline{a},w_A}_{L,V}(S)$
14. for $i = 1$ to $m - 1$
15.     $(S, C[i]) \leftarrow \rho(S, M[i])$
16.     $S \leftarrow \mathsf{ICEnc}^{\overline{a+i},2}_{L,V}(S)$
17. end for
18. $M'[m] \leftarrow \mathtt{pad}_n(M[m])$
19. $(S, C'[m]) \leftarrow \rho(S, M'[m])$
20. $S \leftarrow \mathsf{ICEnc}^{\overline{a+m},w_M}_{L,V}(S)$
21. $C[m] \leftarrow \mathtt{lmt}_{|M[m]|}(C'[m])$
22. $(\eta, T) \leftarrow \rho(S, 0^n)$
23. $C \leftarrow C[1] \| C[2] \| \ldots \| C[m]$
24. return $(C, T)$
```

```
Algorithm $\mathsf{Remus\text{-}N.Dec}_K(N, A, C, T)$
 1. $(L, V) \leftarrow \mathsf{KDF}(N, K)$
 2. $S \leftarrow 0^n$
 3. $(A[1], \ldots, A[a]) \xleftarrow{n} A$
 4. $(C[1], \ldots, C[m]) \xleftarrow{n} C$
 5.  if $|A[a]| < n$ then $w_A \leftarrow 13$ else $12$
 6.  if $|C[m]| < n$ then $w_C \leftarrow 11$ else $10$
 7. $A[a] \leftarrow \mathtt{pad}_n(A[a])$
 8. for $i = 1$ to $a - 1$
 9.     $(S, \eta) \leftarrow \rho(S, A[i])$
10.     $S \leftarrow \mathsf{ICEnc}^{\overline{i},4}_{L,V}(S)$
11. end for
12. $(S, \eta) \leftarrow \rho(S, A[a])$
13. $S \leftarrow \mathsf{ICEnc}^{\overline{a},w_A}_{L,V}(S)$
14. for $i = 1$ to $m - 1$
15.     $(S, M[i]) \leftarrow \rho^{-1}(S, C[i])$
16.     $S \leftarrow \mathsf{ICEnc}^{\overline{a+i},2}_{L,V}(S)$
17. end for
18. $\widetilde{S} \leftarrow (0^{|C[m]|} \| \mathtt{rmt}_{n-|C[m]|}(G(S)))$
19. $C'[m] \leftarrow \mathtt{pad}_n(C[m]) \oplus \widetilde{S}$
20. $(S, M'[m]) \leftarrow \rho^{-1}(S, C'[m])$
21. $M[m] \leftarrow \mathtt{lmt}_{|C[m]|}(M'[m])$
22. $S \leftarrow \mathsf{ICEnc}^{\overline{a+m},w_C}_{L,V}(S)$
23. $(\eta, T^*) \leftarrow \rho(S, 0^n)$
24. $M \leftarrow M[1] \| M[2] \| \ldots \| M[m]$
25. if $T^* = T$ then return $M$ else $\perp$
```

```
Algorithm $\rho(S, M)$
 1. $C \leftarrow M \oplus G(S)$
 2. $S' \leftarrow S \oplus M$
 3. return $(S', C)$
```

```
Algorithm $\rho^{-1}(S, C)$
 1. $M \leftarrow C \oplus G(S)$
 2. $S' \leftarrow S \oplus M$
 3. return $(S', M)$
```

Figure 6: Encryption and decryption of Remus-N. It uses TBC ICE consisting of KDF and ICEnc. Lines of [**if (statement) then** $X \leftarrow x$ **else** x'] are shorthand for [**if (statement) then** $X \leftarrow x$ **else** $X \leftarrow x'$]. The dummy variable η is always discarded. Remus-N1 is used as a working example. Depending on the Remus-N version, an appropriate variant of ICE will be used.

and decryption queries σ_{auth}, and time complexity t_B. Then

$$\mathbf{Adv}^{\mathtt{priv}}_{\mathsf{Romulus\text{-}N}}(\mathcal{A}) \leqslant \mathbf{Adv}^{\mathtt{tprp}}_{\widetilde{E}}(\mathcal{A}'), \tag{1}$$

$$\mathbf{Adv}^{\mathtt{auth}}_{\mathsf{Romulus\text{-}N}}(\mathcal{B}) \leqslant \mathbf{Adv}^{\mathtt{tprp}}_{\widetilde{E}}(\mathcal{B}') + \frac{3q_d}{2^n} + \frac{2q_d}{2^\tau}$$

hold for some $(\sigma_{priv}, t_A + O(\sigma_{priv}))$-TPRP adversary $\mathcal{A}'$, and for some $(\sigma_{auth}, t_B + O(\sigma_{auth}))$-TPRP adversary $\mathcal{B}'$.

Theorem 1 holds for all the members of Romulus-N. Although the bounds of Theorem 1 look as if the adversary has almost no limitation in query complexity except q_d, there will be inherent ones from the specification, say the maximum input length. This holds for all the security claims we make.

4.1.2 Security of Romulus-M

For an encryption query (N, A, M), the number of effective blocks is $\lfloor a/2 \rfloor + \lfloor m/2 \rfloor + 2 + m'$, where $(A[1], \ldots, A[a]) \xleftarrow{n,t} A$, $(M[1], \ldots, M[m]) \xleftarrow{n,t} M$ (or $(M[1], \ldots, M[m]) \xleftarrow{t,n} M$), and $(M[1], \ldots, M[m']) \xleftarrow{n} M$. For a decryption query (N, A, C, T), it is similarly defined by $(C[1], \ldots, C[m]) \xleftarrow{n,t} C$ or $(C[1], \ldots, C[m]) \xleftarrow{t,n} C$, and $(C[1], \ldots, C[m']) \xleftarrow{n} C$.

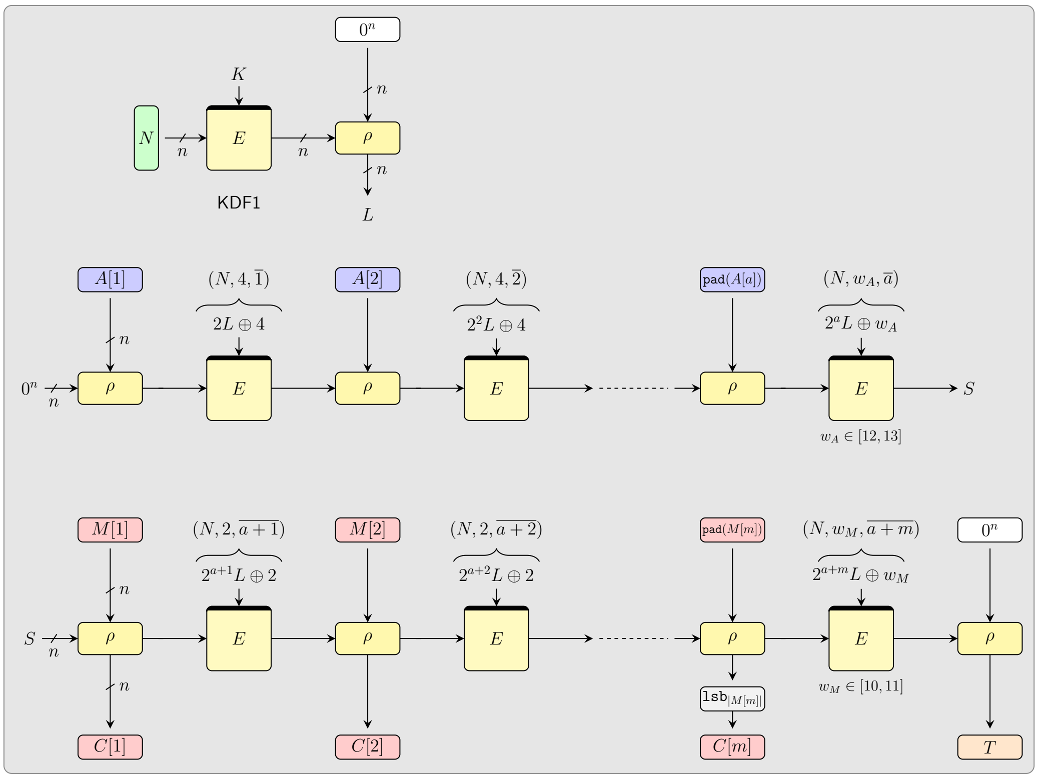


Figure 7: Remus-N with ICE1 (Remus-N1). (Top) Key derivation. (Middle) Processing of AD. (Bottom) Encryption. The domain separation B being of 8 bits only, $\oplus B$ is to be interpreted as $\oplus 0^{120} \| B$.

Theorem 2. *Let $\mathcal{A}$ be an NR privacy adversary against* Romulus-M *that uses q_e encryption queries with time complexity t_A and with total number of effective blocks σ_{priv}. Let $\mathcal{B}$ be an NR authenticity adversary against* Romulus-M *using q_e encryption queries and q_d decryption queries, with total number of effective blocks for encryption and decryption queries σ_{auth} and with time complexity t_B. Then we have*

$$\mathbf{Adv}^{\mathtt{priv}}_{\mathsf{Romulus\text{-}M}}(\mathcal{A}) \leqslant \mathbf{Adv}^{\mathtt{tprp}}_{\tilde{E}}(\mathcal{A}'),$$
$$\mathbf{Adv}^{\mathtt{auth}}_{\mathsf{Romulus\text{-}M}}(\mathcal{B}) \leqslant \mathbf{Adv}^{\mathtt{tprp}}_{\tilde{E}}(\mathcal{B}') + \frac{5q_d}{2^n}$$

for some $(\sigma_{priv}, t_A + O(\sigma_{priv}))$-TPRP adversary $\mathcal{A}'$, and $(\sigma_{auth}, t_B + O(\sigma_{auth}))$-TPRP adversary $\mathcal{B}'$.

Theorem 3. *Let $\mathcal{A}$ be an NM privacy adversary and let $\mathcal{B}$ be an NM authenticity adversary, both are against* Romulus-M, *that use the same parameters as in Theorem 2, and that can repeat a nonce at most $1 \leqslant r \leqslant 2^{n-1}$ times in encryption queries. Then we have*

$$\mathbf{Adv}^{\mathtt{nm\text{-}priv}}_{\mathsf{Romulus\text{-}M}}(\mathcal{A}) \leqslant \mathbf{Adv}^{\mathtt{tprp}}_{\tilde{E}}(\mathcal{A}') + \frac{4r\sigma_{\mathrm{priv}}}{2^n},$$
$$\mathbf{Adv}^{\mathtt{nm\text{-}auth}}_{\mathsf{Romulus\text{-}M}}(\mathcal{B}) \leqslant \mathbf{Adv}^{\mathtt{tprp}}_{\tilde{E}}(\mathcal{B}') + \frac{4rq_e + 5rq_d}{2^n}$$

for some $(\sigma_{priv}, t_A + O(\sigma_{priv}))$-TPRP adversary $\mathcal{A}'$ and $(\sigma_{auth}, t_B + O(\sigma_{auth}))$-TPRP adversary $\mathcal{B}'$.

We note that it is possible to unify Theorems 2 and 3 into a single theorem statement,

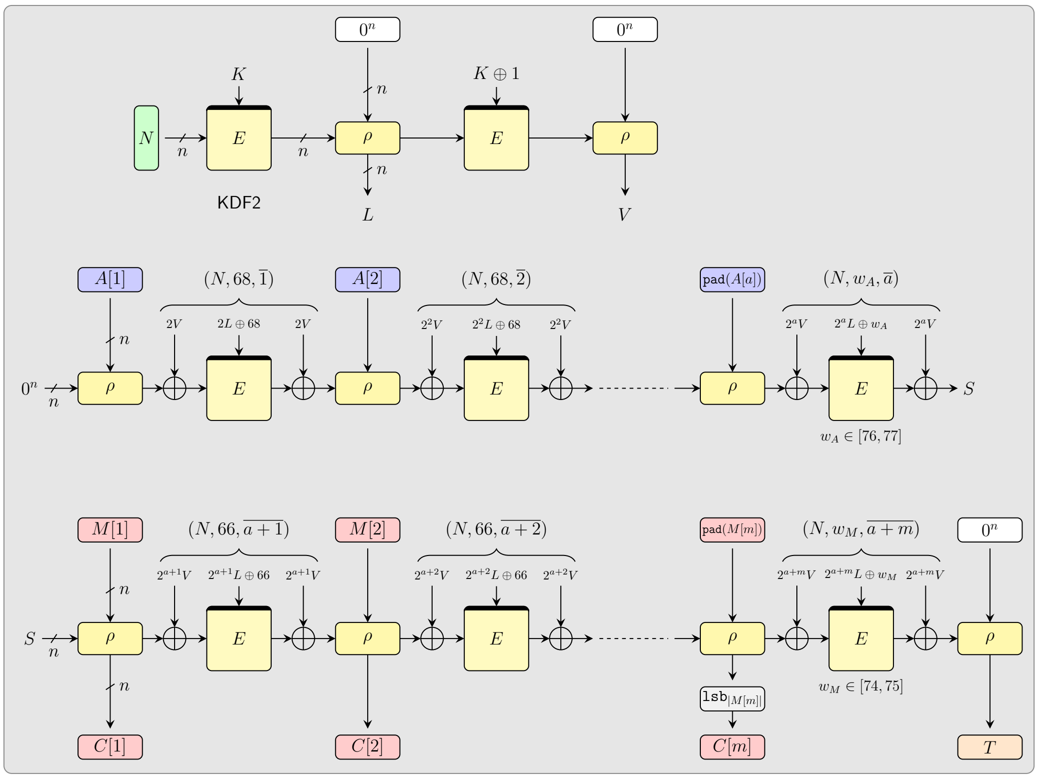


Figure 8: Remus-N with ICE2 (Remus-N2). (Top) Key derivation. (Middle) Processing of AD. (Bottom) Encryption. The domain separation B being of 8 bits only, $\oplus B$ is to be interpreted as $\oplus 0^{120} \| B$.

while we chose to differentiate them for the difference of the security model, and separating the proofs makes them more transparent.

Bit Security. Both Romulus-N and Romulus-M have n-bit security for NR-privacy and NR-authenticity, and Romulus-M has $n/2$-bit security for NM-privacy and NM-authenticity, with graceful degradation in the number of nonce repetitions [PS16], i.e., a small number of nonce repetitions do not harm the security too much. The bounds of Romulus-N is the same as ΘCB3 except the tiny difference in their constants.

4.2 Security of Remus

Unlike Romulus, the security of Remus relies on the Ideal-Cipher Model (ICM), that is, the underlying $E : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{M}$ is sampled uniformly over all the block ciphers, and can be queried at any time for both directions: $\widehat{C} \leftarrow E(\widehat{K}, \widehat{M})$ or $\widehat{M} \leftarrow E^{-1}(\widehat{K}, \widehat{C})$, where the key input $\widehat{K}$ is a part of the query. A query to E is called a primitive query. An ordinary (encryption/decryption) query to Remus-N or Remus-M may be called a construction query. The framework is similar to ICM-based TBC constructions, e.g. Mennink [Men15].

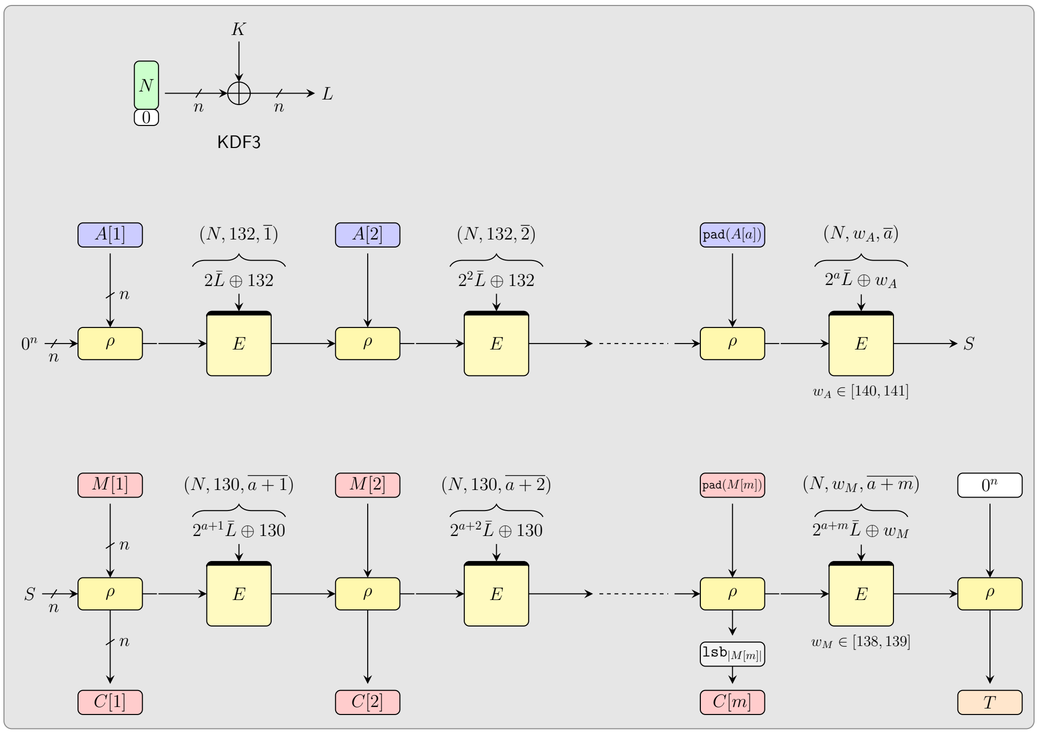


Figure 9: Remus-N with ICE3 (Remus-N3). (Top) Key derivation (computing $(2^D L[1])||(B \oplus L[2])$, we use the $\bar{L}$ notation to represent the fact that the multiplication is only done on $L[1]$, the first $k-8$ bits of L). (Middle) Processing of AD. (Bottom) Encryption. The domain separation B being of 8 bits only, $\oplus B$ is to be interpreted as $\oplus\, 0^{120} \| B$.

4.2.1 Security of Remus-N

For $A \in \{0,1\}^*$, we say A has a AD blocks if $|A|_n = a$. Similarly, for plaintext $M \in \{0,1\}^*$ we say M has m message blocks if $|M|_n = m$. The same holds for the ciphertext C. For the encryption query (N, A, M) or decryption query (N, A, C, T) of a AD blocks and m message blocks, the total number of TBC calls is at most $a + m$, which is called the number of *effective blocks* of a query. As before $\tau \in [\![n]\!]$ is the tag length.

Theorem 4. *Let $\mathcal{A}$ be an NR privacy adversary against* Remus-N *using q_e encryption (construction) queries and q_p primitive queries with total number of effective blocks in encryption queries σ_{priv}. Moreover, let $\mathcal{B}$ be an NR authenticity adversary against* Remus-N *using q_e encryption queries and q_d decryption queries, with total number of effective blocks*

```
Algorithm Remus-M.Enc_K(N, A, M)
 1. $(L, V) \leftarrow \mathsf{KDF}(N, K)$
 2. $S \leftarrow 0^n$
 3. $(A[1], \ldots, A[a]) \xleftarrow{n} A$
 4. $(M[1], \ldots, M[m]) \xleftarrow{n} M$
 5. if $|A[a]| < n$ then $w_A \leftarrow 45$ else $44$
 6. if $|M[m]| < n$ then $w_M \leftarrow 47$ else $46$
 7. $A[a] \leftarrow \mathtt{pad}_n(A[a])$
 8. for $i = 1$ to $a - 1$
 9.     $(S, \eta) \leftarrow \rho(S, A[i])$
10.     $S \leftarrow \mathsf{ICEnc}_{L,V}^{\overline{i},36}(S)$
11. end for
12. $(S, \eta) \leftarrow \rho(S, A[a])$
13. $S \leftarrow \mathsf{ICEnc}_{L,V}^{\overline{a},w_A}(S)$
14. for $i = 1$ to $m - 1$
15.     $(S, \eta) \leftarrow \rho(S, M[i])$
16.     $S \leftarrow \mathsf{ICEnc}_{L,V}^{\overline{a+i},38}(S)$
17. end for
18. $M'[m] \leftarrow \mathtt{pad}_n(M[m])$
19. $(S, \eta) \leftarrow \rho(S, M'[m])$
20. $S \leftarrow \mathsf{ICEnc}_{L,V}^{\overline{a+m},w_M}(S)$
21. $(\eta, T) \leftarrow \rho(S, 0^n)$
22. if $M = \epsilon$ then return $(\epsilon, T)$
23. $S \leftarrow T$
24. for $i = 1$ to $m - 1$
25.     $S \leftarrow \mathsf{ICEnc}_{L,V}^{\overline{i-1},34}(S)$
26.     $(S, C[i]) \leftarrow \rho(S, M[i])$
27. end for
28. $S \leftarrow \mathsf{ICEnc}_{L,V}^{\overline{m-1},34}(S)$
29. $(\eta, C'[m]) \leftarrow \rho(S, M'[m])$
30. $C[m] \leftarrow \mathtt{lsb}_{|M[m]|}(C'[m])$
31. $C \leftarrow C[1] \,\|\, C[2] \,\|\, \ldots \,\|\, C[m]$
32. return $(C, T)$
```

```
Algorithm Remus-M.Dec_K(N, A, C, T)
 1. $(L, V) \leftarrow \mathsf{KDF}(N, K)$
 2. if $C = \epsilon$ then $M \leftarrow \epsilon$
 3. else
 4.     $S \leftarrow T$
 5.     $(C[1], \ldots, C[m]) \xleftarrow{n} C$
 6.     $z \leftarrow |C[m]|$
 7.     $C[m] \leftarrow \mathtt{pad}_n(C[m])$
 8.     for $i = 1$ to $m$
 9.         $S \leftarrow \mathsf{ICEnc}_{L,V}^{\overline{i-1},34}(S)$
10.         $(S, M[i]) \leftarrow \rho^{-1}(S, C[i])$
11.     end for
12.     $M[m] \leftarrow \mathtt{lsb}_z(M[m])$
13.     $M \leftarrow M[1] \,\|\, \ldots \,\|\, M[m]$
14. $S \leftarrow 0^n$
15. $(A[1], \ldots, A[a]) \xleftarrow{n} A$
16. if $|A[a]| < n$ then $w_A \leftarrow 45$ else $44$
17. if $|M[m]| < n$ then $w_M \leftarrow 47$ else $46$
18. $A[a] \leftarrow \mathtt{pad}_n(A[a])$
19. for $i = 1$ to $a - 1$
20.     $(S, \eta) \leftarrow \rho(S, A[i])$
21.     $S \leftarrow \mathsf{ICEnc}_{L,V}^{\overline{i},36}(S)$
22. end for
23. $(S, \eta) \leftarrow \rho(S, A[a])$
24. $S \leftarrow \mathsf{ICEnc}_{L,V}^{\overline{a},w_A}(S)$
25. for $i = 1$ to $m - 1$
26.     $(S, \eta) \leftarrow \rho(S, M[i])$
27.     $S \leftarrow \mathsf{ICEnc}_{L,V}^{\overline{a+i},38}(S)$
28. end for
29. $M'[m] \leftarrow \mathtt{pad}_n(M[m])$
30. $(S, \eta) \leftarrow \rho(S, M'[m])$
31. $S \leftarrow \mathsf{ICEnc}_{L,V}^{\overline{a+m},w_M}(S)$
32. $(\eta, T^*) \leftarrow \rho(S, 0^n)$
33. if $T^* = T$ then return $M$ else $\perp$
```

```
Algorithm ρ(S, M)
 1. $C \leftarrow M \oplus G(S)$
 2. $S' \leftarrow S \oplus M$
 3. return $(S', C)$
```

```
Algorithm ρ^{-1}(S, C)
 1. $M \leftarrow C \oplus G(S)$
 2. $S' \leftarrow S \oplus M$
 3. return $(S', M)$
```

Figure 10: Encryption and decryption of Remus-M. Remus-M1 is used as a working example.Depending on the Remus-M version, an appropriate variant of ICE will be used.

for encryption and decryption queries σ_{auth}, and q_p primitive queries. Then we have

$$\mathbf{Adv}^{\mathtt{priv}}_{\mathsf{Remus\text{-}N1}}(\mathcal{A}) \leqslant \frac{9\sigma^2_{\text{priv}} + 4q_p\sigma_{\text{priv}}}{2^n} + \frac{2q_p}{2^n},$$

$$\mathbf{Adv}^{\mathtt{priv}}_{\mathsf{Remus\text{-}N2}}(\mathcal{A}) \leqslant \frac{9\sigma^2_{\text{priv}} + 4q_p\sigma_{\text{priv}}}{2^{2n}} + \frac{2q_p}{2^n},$$

$$\mathbf{Adv}^{\mathtt{priv}}_{\mathsf{Remus\text{-}N3}}(\mathcal{A}) \leqslant \frac{0.5\sigma^2_{\text{priv}}}{2^{k-8}} + \frac{q_p\sigma_{\text{priv}}}{2^k},$$

$$\mathbf{Adv}^{\mathtt{auth}}_{\mathsf{Remus\text{-}N1}}(\mathcal{B}) \leqslant \frac{9\sigma^2_{\text{auth}} + 4q_p\sigma_{\text{auth}}}{2^n} + \frac{2q_p}{2^n} + \frac{3q_d}{2^n} + \frac{2q_d}{2^\tau},$$

$$\mathbf{Adv}^{\mathtt{auth}}_{\mathsf{Remus\text{-}N2}}(\mathcal{B}) \leqslant \frac{9\sigma^2_{\text{auth}} + 4q_p\sigma_{\text{auth}}}{2^{2n}} + \frac{2q_p}{2^n} + \frac{3q_d}{2^n} + \frac{2q_d}{2^\tau},$$

$$\mathbf{Adv}^{\mathtt{auth}}_{\mathsf{Remus\text{-}N3}}(\mathcal{B}) \leqslant \frac{0.5\sigma^2_{\text{auth}}}{2^{k-8}} + \frac{q_p\sigma_{\text{auth}}}{2^k} + \frac{3q_d}{2^n} + \frac{2q_d}{2^\tau}.$$

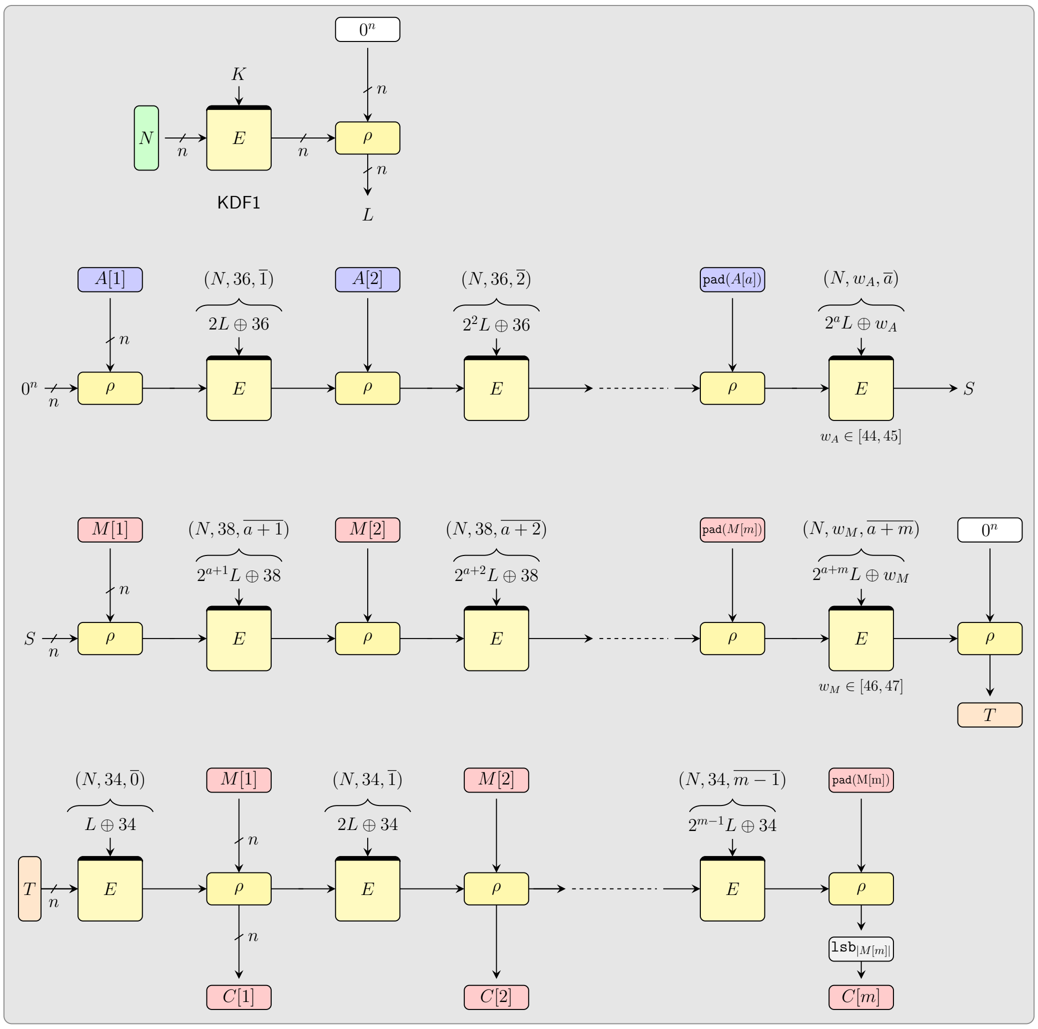


Figure 11: Remus-M with ICE1 (Remus-M1). (Top) Key derivation. (Middle-Top) Processing of AD (Middle-Bottom) Processing of M authentication (Bottom) Encryption. The domain separation B being of 8 bits only, $\oplus B$ is to be interpreted as $\oplus \mathsf{0}^{120} \parallel B$.

4.2.2 Security of Remus-M

For the encryption query (N, A, M) or decryption query (N, A, C, T) of a AD blocks and m message blocks, the number of total TBC calls is at most $a + 2m$, which is called the number of *effective blocks* of a query.

Theorem 5. *Let $\mathcal{A}$ be an NR privacy adversary against* Remus-M *using q_e encryption (construction) queries and q_p primitive queries with total number of effective blocks in encryption queries σ_{priv}. Moreover, let $\mathcal{B}$ be an NR authenticity adversary using q_e encryption queries and q_d decryption queries, with total number of effective blocks for*

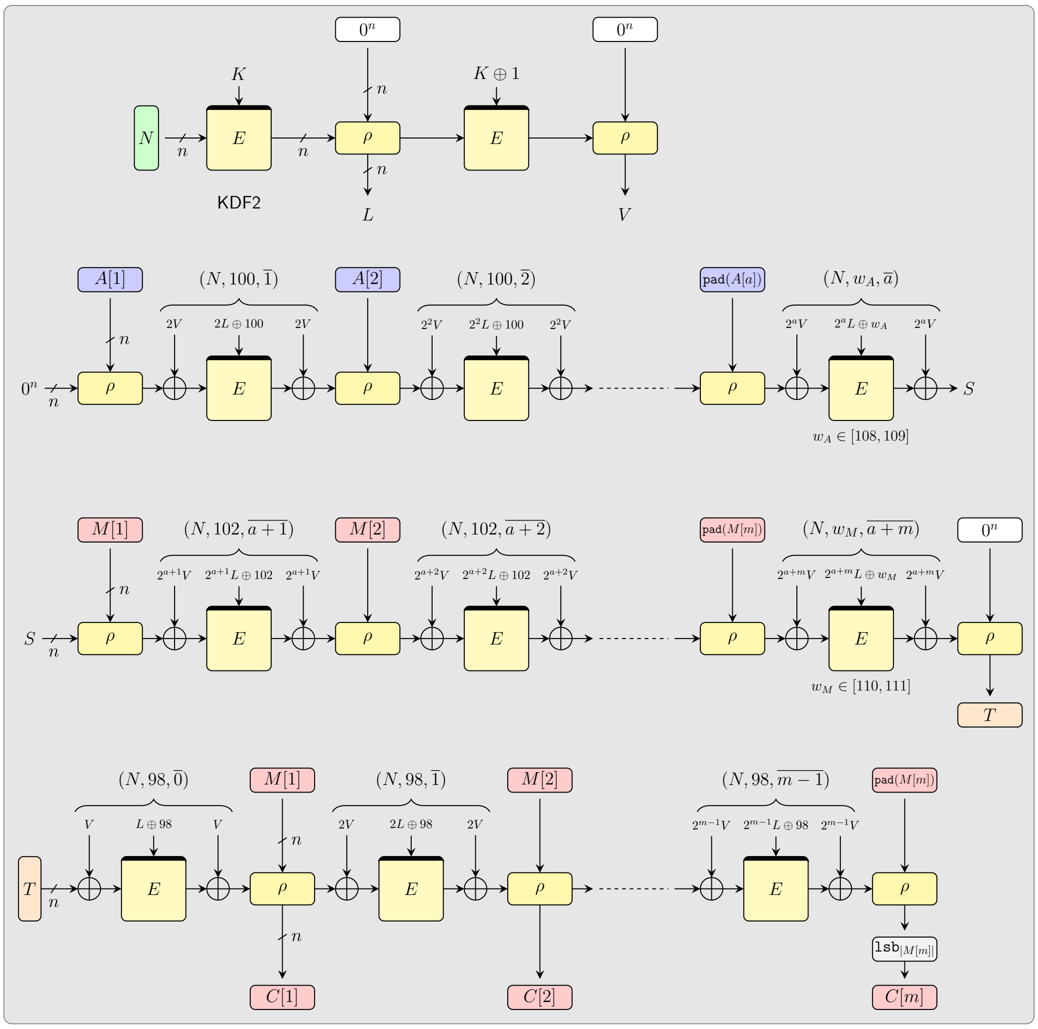


Figure 12: Remus-M with ICE2 (Remus-M2). (Top) Key derivation. (Middle-Top) Processing of AD (Middle-Bottom) Processing of M authentication (Bottom) Encryption. The domain separation B being of 8 bits only, $\oplus B$ is to be interpreted as $\oplus 0^{120} \| B$.

encryption and decryption queries σ_{auth}, and q_p primitive queries. Then we have

$$\mathbf{Adv}^{\mathtt{priv}}_{\mathsf{Remus\text{-}M1}}(\mathcal{A}) \leqslant \frac{9\sigma^2_{\mathrm{priv}} + 4q_p\sigma_{\mathrm{priv}}}{2^n} + \frac{2q_p}{2^n},$$

$$\mathbf{Adv}^{\mathtt{priv}}_{\mathsf{Remus\text{-}M2}}(\mathcal{A}) \leqslant \frac{9\sigma^2_{\mathrm{priv}} + 4q_p\sigma_{\mathrm{priv}}}{2^{2n}} + \frac{2q_p}{2^n},$$

$$\mathbf{Adv}^{\mathtt{auth}}_{\mathsf{Remus\text{-}M1}}(\mathcal{B}) \leqslant \frac{9\sigma^2_{\mathrm{auth}} + 4q_p\sigma_{\mathrm{auth}}}{2^n} + \frac{2q_p}{2^n} + \frac{5q_d}{2^n},$$

$$\mathbf{Adv}^{\mathtt{auth}}_{\mathsf{Remus\text{-}M2}}(\mathcal{B}) \leqslant \frac{9\sigma^2_{\mathrm{auth}} + 4q_p\sigma_{\mathrm{auth}}}{2^{2n}} + \frac{2q_p}{2^n} + \frac{5q_d}{2^n}.$$

Theorem 6. *Let $\mathcal{A}$ be an NM privacy adversary and let $\mathcal{B}$ be an NM authenticity adversary, both are against* Remus-M*, that use the parameters as specified in Theorem 5, and can* *repeat a nonce at most $1 \leqslant r \leqslant 2^{n-1}$ times in encryption queries. Then we have*

$$\mathbf{Adv}^{\texttt{nm-priv}}_{\textsf{Remus-M1}}(\mathcal{A}) \leqslant \frac{9\sigma^2_{\text{priv}} + 4q_p\sigma_{\text{priv}}}{2^n} + \frac{2q_p}{2^n} + \frac{4r\sigma_{\text{priv}}}{2^n},$$
$$\mathbf{Adv}^{\texttt{nm-priv}}_{\textsf{Remus-M2}}(\mathcal{A}) \leqslant \frac{9\sigma^2_{\text{priv}} + 4q_p\sigma_{\text{priv}}}{2^{2n}} + \frac{2q_p}{2^n} + \frac{4r\sigma_{\text{priv}}}{2^n},$$
$$\mathbf{Adv}^{\texttt{nm-auth}}_{\textsf{Remus-M1}}(\mathcal{B}) \leqslant \frac{9\sigma^2_{\text{auth}} + 4q_p\sigma_{\text{auth}}}{2^n} + \frac{2q_p}{2^n} + \frac{4rq_e}{2^n} + \frac{5rq_d}{2^n},$$
$$\mathbf{Adv}^{\texttt{nm-auth}}_{\textsf{Remus-M2}}(\mathcal{B}) \leqslant \frac{9\sigma^2_{\text{auth}} + 4q_p\sigma_{\text{auth}}}{2^{2n}} + \frac{2q_p}{2^n} + \frac{4rq_e}{2^n} + \frac{5rq_d}{2^n}.$$

We note that, as with the case of Romulus-M, Theorems 5 and 6 can be unified. We separate them for the difference of the security model and clarity of the proofs.

Bit Security. For NAE variants, Remus-N1 and Remus-N2 have $n/2$-bit and n-bit security, while Remus-N3 has $\min\{(k-8)/2, n\}$-bit security, for NR-privacy and NR-authenticity. For MRAE variants, Remus-M1 has $n/2$-bit security for all relevant notions (NR/NM-privacy and authenticity), and Remus-M2 has n-bit security for NR-notions and $n/2$-bit security for NM-notions with a graceful degradation with respect to nonce repetition in a similar manner to Romulus-M.

4.3 Remarks

All the security bounds hold as long as G is sound. As explained earlier, it is also straightforward to extend them to inputs of any bit strings rather than bytes (see Section 3.1). For all schemes, the basic proof strategy is similar to the proofs of iCOFB [CIMN17b], and the recent proposal of Naito and Sugawara's PFB mode [NS20], but we needed a careful case analysis.

The Use of ICM. ICM has been acknowledged as a meaningful security proof model, especially in the field of hash function constructions. As described in the introduction, we can also obtain standard model proofs for Remus, by assuming the intermediate TBC ICE as a keyed primitive called tweakable pseudorandom permutation (TPRP). The standard model proofs are actually a part of ICM proofs. Specifically, the security bounds under this standard model will appear in the hybrid argument of ICM proofs. A similar technique appeared in some permutation-based schemes [MMH$^+$14, CDH$^+$12, MRV15], where the keyed primitive is a variant of the Even-Mansour cipher. We warn that the assumption that ICE is a TPRP does not imply its perfect security: there are generic attacks which work even if the underlying TBC is an ideal-cipher. Therefore, the expected bit security levels are identical for both ICM and this standard model analyses, and we only present the security bounds under ICM.

Tightness of the Security Bounds. We note that our security bounds are tight. Indeed, the existing third-party analysis [Mej19] for Remus-N3 and [DJN19] for Remus-N1 and Remus-M1 show their tightness. We emphasize that these analysis do not contradict our security bounds.

5 Design Rationale of Romulus and Remus

Rationale of NAE Mode. Romulus-N has a similar structure as a mode called iCOFB, which appeared in the full version of CHES 2017 paper [CIMN17b]. Because it was introduced to show the feasibility of the main proposal of [CIMN17a], block cipher mode COFB, it does not work as a full-fledged AE using conventional TBCs. Therefore, starting from iCOFB, we apply numerous changes for improving efficiency while achieving high security. As a result, Romulus-N becomes a much more advanced, sophisticated NAE mode based on a TBC. The security bound of Romulus-N is essentially equivalent to ΘCB3, having full n-bit security, since the bounds only differ in their constants.

Recently, a new mode called PFB was published by Naito and Sugawara [NS20], which holds some similarities and design goals to Romulus-N. It uses different feedback functions for the associated data, plaintext and ciphertext, with the purpose of providing partial parallelizability for the encryption algorithm, where the associated data processing and the ciphertext decryption is serial in nature, while the plaintext encryption can be parallelized. Since our goal in Romulus and Remus is to achieve very lightweight hardware implementations, we chose a fixed feedback function for all the parts of the algorithm, in order to avoid additional multiplexers, given that applications of such partial parallelizability are limited. Moreover, it was shown during the CAESAR competition that the parallelized hardware implementations of block-cipher based modes, e.g. Deoxys, does not provide huge performance gains as in the case of software implementations [KCP17].

By seeing Remus-N as a mode of a TBC (ICE), Remus-N also has a similar structure as iCOFB [CIMN17b]. Assuming ICE is an ideally secure TBC, the security bound of Remus-N is also essentially equivalent to ΘCB3, having full n-bit security. The remaining problem is how to efficiently instantiate a TBC. We could use a dedicated TBC, or a conventional block cipher mode (such as XEX [Rog04a]), however, they have certain limitations on security and efficiency. To overcome such limitations, we choose to use a block cipher with tweak-dependent key/mask derivation. This approach, initiated by Mennink [Men15], improves the performance over earlier approaches, at the expense of the ideal-cipher model for security. Specifically, the TBC ICE has three variants, where ICE1 and ICE2 can be seen as a variant of XHX [JLM+17], and ICE3 is a more classical one combined with a doubling mask [Rog04a]. Each variant has its own security level, namely, ICE1 has $n/2$-bit security, ICE2 has n-bit security, and ICE3 has $(n/2-4)$-bit security. They have different computation cost for key/mask derivations and have different state sizes. Given the n-bit security of *outer* TBC-based mode, the standard hybrid argument shows that the security of Remus-N is effectively determined by the security of the internal ICE.

Rationale of MRAE Mode. Romulus-M and Remus-M are designed as MRAE modes following the structure of SIV [RS06] and SCT [PS16]. Romulus-M reuses the components of Romulus-N and Remus-M reuses the components of Remus-N as much as possible to inherit their implementation advantages and the security. In fact, this brings us several advantages (not only for implementation aspects) over SIV/SCT. Compared with SCT, Romulus-M needs fewer primitive calls because of the faster MAC. Moreover, Romulus-M has a smaller state than SCT because of the single-state encryption part taken from Romulus-N (SCT employs a variant of counter mode). Remus-M needs an equivalent number of primitive calls as SCT. The difference is in the primitive: Remus-M uses an n-bit block cipher while SCT uses an n-bit-block dedicated TBC. Moreover, Remus-M has a smaller state than SCT because of the single-state encryption part taken from Remus-N.

The provable security of Romulus-M is similar to SCT: the security depends on the maximum number of repetition of a nonce in encryption (r), and if $r = 1$ (i.e., NR adversary) we have the full n-bit security. Security will gradually decrease as r increases, also known as "graceful degradation", and even if r equals the number of encryption queries, implying nonces are fixed, we maintain the birthday-bound, $n/2$-bit security. Similarly to Remus-N, the provable security of Remus-M is effectively determined by the internal ICE. For Remus-M2, thanks to n-bit security of ICE2, its security is also similar to SCT. For NR-adversaries, we have the full n-bit security. Security will gradually decrease as r increases, and even if r equals the number of encryption queries, we maintain the birthday-bound, $n/2$-bit security. For Remus-M1, the security is $n/2$ bits for both NR and NM adversaries due to the $n/2$-bit security of ICE1.

ZAE [IMPS17] is another TBC-based MRAE. Although it is faster than SCT, the state size is much larger than SCT, Romulus-M and Remus-M.

Theoretical Efficiency Comparison. In Table 1, we compare Romulus-N and Remus-N to ΘCB3 and a group of recently proposed lightweight AEAD modes. In the table, state size is the minimum number of bits that the mode has to maintain during its operation, and rate is the ratio of input data length divided by the total output length of the primitive needed to process that input. ΘCB3 is a well-studied TBC-based AEAD mode. COFB is a BC-based lightweight AEAD mode. Beetle is a Sponge-based AEAD mode, but it holds a lot of resemblance to Remus-N. The comparison follows the following guidelines, while trying to be fair in comparing designs that follow completely different approaches:

1. $k = 128$ for all the designs.
2. n is the input block size (in bits) for each primitive call.
3. λ is the security level of the design.
4. For BC/TBC based designs, the key is considered to be stored inside the design, but we also consider that the encryption and decryption keys are interchangeable, i.e., the encryption key can be derived from the decryption key and vice versa. Hence, there is no need to store the master key in additional storage. The same applies for the nonce.
5. For Sponge and Sponge-like designs, if the key/nonce are used only during initialization, then they are counted as part of the state and do not need extra storage. However, in designs like Ascon, where the key is used again during finalization, we assume the key storage is part of the state, as the key should be supplied only once as an input.

Our comparative analysis shows that Romulus-N is smaller and more efficient than ΘCB3 for the same security level. Moreover, the cost of processing AD is about half that of the message. For example, in the case of Romulus-N1, if the message and AD have equal length, there is an extra speed up of ~ 1.33x, which means that the efficiency even increases from 3.5λ to 2.625λ, compared to 4.5λ in case of ΘCB3, which makes Romulus-N a very promising candidate for NAE, for both short and long messages. Besides, it shows that Remus-N achieves its goals, as Remus-N1 has a $2n$-bit state, which is smaller than COFB and equal to Beetle. Remus-N1 and COFB both have birthday security, i.e., $n/2$. Beetle achieves higher security, at the expense of using a $2n$-bit permutation. Our analysis also shows that among the considered AEAD modes, Remus-N2 achieves the lowest R/S ratio, with a state size of $3n$ but only an n-bit permutation. Since Remus-N3 uses a 64-bit block cipher, we manage to achieve very small area and more relaxed state size.

A similar comparison is shown in Table 2 for Misuse-Resistant BC- and TBC-based AEAD modes. It shows that Remus-M2 is particularly very efficient. Also, Romulus-M is very efficient. Not only the state size is smaller, but also it is faster. For example, Romulus-M1 is 25% faster (1.33x speed-up) than SCT for the same parameters, when $|A| = 0$, and it is even faster when $|A| > 0$.

6 Hardware Implementations

6.1 General Architecture and Hardware Estimates

The goal of the design of Romulus and Remus is to have a very small area overhead over the underlying TBC, specially for the round-based implementations. In order to achieve this goal, we set two requirements:

1. There should be no extra Flip-Flops over what is already required by the TBC, since Flip-Flops are very costly ($4 \sim 7$ GEs per Flip-Flop).
2. The number of possible inputs to each Flip-Flop and outputs of the circuits have to be minimized. This is in order to reduce the number of multiplexers required, which is usually one of the causes of efficiency reduction between the specification and implementation.

One of the advantages of Skinny as a lightweight TBC is that it has a very simple datapath, consisting of a simple state register followed by a low-area combinational circuit, where the same circuit is used for all the rounds, so the only multiplexer required is to select between the initial input for the first round and the round output afterwards (Figure 13(a)), and it has been shown that this multiplexer can even have lower cost than a normal multiplexer if it is combined with the Flip-Flops by using Scan-Flops (Figure 13(b)) [JMPS17]. However, when used inside an AEAD mode, challenges arise, such as how to store the key and nonce, as the key scheduling algorithm will change these values after each block encryption. The same goes for the block counter. In order to avoid duplicating the storage elements for these values; one set to be used to execute the TBC and one set to be used by the mode to maintain the current value, we studied the relation between the original and final value of the tweakey. Since the key scheduling algorithm of Skinny is fully linear and has very low area (most of the algorithm is just routing and renaming of different bytes), the full algorithm can be inverted using a very small circuit that costs 64 XOR gates for Romulus-N1 and 0 gates for Remus. Moreover, the LFSR computation required between blocks can be implemented on top of this circuit, costing 3 extra XOR gates. This operation can be computed in parallel to ρ, such that when the state is updated for the next block, the tweakey key required is also ready. This costs only ~ 67 XOR gates as opposed to ~ 320 Flip-Flops that will, otherwise, be needed to maintain the tweakey value. Hence, the mode was designed with the architecture in Figure 13(b) in mind, where only a full-width state-register is used, carrying the TBC state and tweakey values, and every cycle, it is either kept without change, updated with the TBC round output (which includes a single round of the key scheduling algorithm) or the output of a simple linear transformation, which consists of ρ/ρ^{-1}, the unrolled inverse key schedule and the block counter. In order estimate the hardware cost of Romulus-N1 the mode we consider the round based implementation with an $n/4$-bit input/output bus:

- 4 XOR gates for computing G.
- 64 XOR gates for computing ρ.
- 67 XOR gates for the correction of the tweakey and counting.
- 56 multiplexers to select whether to choose to increment the counter or not.
- 320 multiplexers to select between the output of the Skinny round and lt.

This adds up to 135 XOR gates and 376 multiplexers. For estimation purposes assume an XOR gate costs 2.25 GEs and a multiplexer costs 2.75 GEs, which adds up to 1337.75 GEs. In the original Skinny paper [BJK+16], the authors reported that Skinny-128-384 requires $4,268$ GEs, which adds up to $\sim 5,605$ GEs. This is ~ 1.4 KGEs smaller than the round based implementation of Ascon [GWDE15]. Moreover, a smart design can make use of the fact that 64 bits of the tweakey of Skinny-128-384 are not used, replacing 64 Flip-Flops by 64 multiplexers reducing an extra ~ 200 GEs. In order to design a combined encryption/decryption circuit, we show below that the decryption costs only extra 32 multiplexers and ~ 32 OR gates, or ~ 100 GEs. A similar analysis is done for Romulus-N2 and Romulus-N3, estmating that they would cost $1,217$ and $1,073$ GEs, respectively, on top of there corresponding Skinny variant, or $5,485$ and $4,385$ GEs, respectively.

These estimations show that Romulus-N is not just competitive theoretically but it can be a very attractive option practically for low area applications. For example, the 8-bit implementation of ACORN, the smallest implementation publicly available for all the round 3 candidates of the CAESAR competition, costs $5,900$ GEs, as shown in [KHYKC17]. If we assume around $\sim 1,000$ GEs as the cost of the CAESAR Hardware API included in that design, as reported in [GWDE15], then Romulus-N3 is still smaller than that. Besides, we believe the area can be even lower using serial implementations of Skinny, which cost $\sim 3,000$ GE for Skinny-128-384 and $\sim 2,000$ GEs for Skinny-128-256, a gain of more than $1,000$ GEs compared to the round-based implementation.

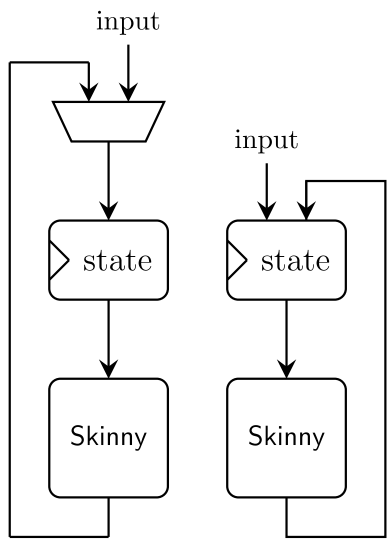
Another possible optimization is to consider the fact that most of the area of Skinny comes from the storage elements, hence, we can speed up Romulus to almost double the speed by using a simple two-round unrolling, which costs $\sim 1,000$ GEs, as only the logic part of Skinny needs replication, which is only $< 20\%$ increase in terms of area.

Romulus-M is estimated to have almost the same area as Romulus-N, except for an additional set of multiplexers in order to use the tag as an initial vector for the encryption part. This indicates that it can be a very lightweight choice for high security applications.

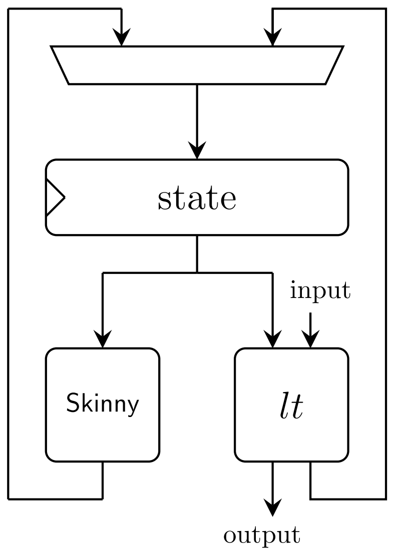
For the serial implementations we followed the currently popular bit-sliding framework [JMPS17] with minor tweaks. The state of Skinny is represented as the Feedback-Shift Register which typically operates on 8 bits at a time, while allowing the 32-bit MixColumns operation, given in Figure 14.

It can be viewed in Figure 14 that several careful design choices such as a lightweight serializable ρ function without the need of any extra storage and a lightweight padding/truncation scheme allow the low area implementations to use a very small number of multiplexers on top of the Skinny circuit for the state update, three 8-bit multiplexer to be exact, two of which have a constant zero input, and ~ 22 XORs for the ρ function and block counter. For the key update functions, we did several experiments on how to serialize the operations and we found the best trade-off is to design a parallel/serial register for every tweakey, where the key schedule and mode operations are done in the same manner of the round based implementation, while the AddRoundKey operation of Skinny is done serial as shown in Figure 14.

Usually there is a disparity between the theoretical estimate of the state size of a mode and the practical implementations. However, our practical implementations show that our theoretical estimates match exactly our implementations for both round based and serial implementations up to a constant term that covers the round constants of Skinny and the Finite State Machine of the API/interface, as shown in Table 3.



(a) Overview of the round based architecture of Skinny.



(b) Overview of the round based architecture of Remus. lt: The linear transformation that includes ρ, block counter and inverse key schedule.

Figure 13: Expected architectures for Skinny and Remus

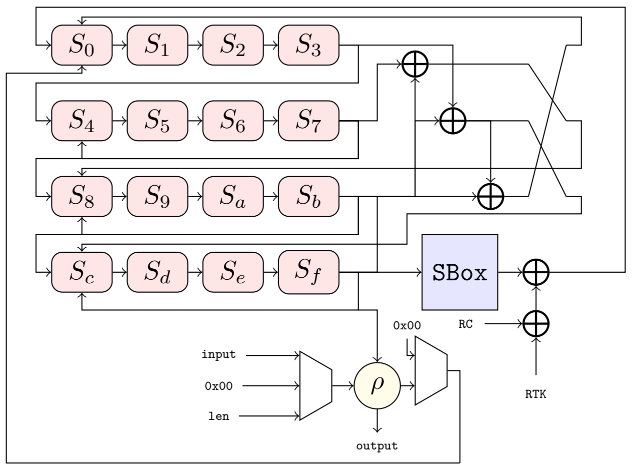


Figure 14: Serial state update function used in Romulus and Remus.

Table 3: Practical state size of Romulus and Remus variants.

Variant	State Size
Romulus-N1	$448 + \mathcal{O}(1)$
Remus-N1	$256 + \mathcal{O}(1)$
Remus-N2	$384 + \mathcal{O}(1)$
Remus-M1	$256 + \mathcal{O}(1)$
Remus-M2	$384 + \mathcal{O}(1)$

6.2 Hardware Implementations Costs

Hardware Cost of Remus-N1**.** The overhead of Remus-N1 is mostly due to the doubling (3 XORs) and ρ operations (68 XORs, assuming the input/output bus has width of 32 bits). Moreover, we need 2 128-bit multiplexers to select the input to the tweakey out of four positive values: K, S (after applying the KDF function), lt, or the Skinny round key. We assume a multiplexer costs $\sim$ 2.75 GEs and an XOR gate costs $\sim$ 2.25 GEs. In total, this adds up to $\sim$ 864 GEs on top of Skinny-128/128. In the original Skinny paper [BJK+16], the authors reported that the round-based implementation of Skinny-128/128 costs $\sim 2,391$ GEs. So we estimate that Remus-N1 should cost $\sim 3,255$ GEs, which is a very small figure, compared to not just TBC based AEAD modes, but in general. For example, ACORN, the smallest CAESAR candidate, costs $\sim 5,900$ GEs. Besides, two further optimizations are applicable. First, we can use the serial Skinny-128/128 implementation, which costs $\sim$ 600 GEs less. The other direction is to unroll Skinny-128/128 to a 2- or 4-round implementation, reducing the number of cycles, at the cost of 1 KGEs per extra round. We do acknowledge that this huge gain in area comes at the cost of reduced security (birthday security). In order to design a combined encryption/decryption circuit, we show below that the decryption costs only extra 32 multiplexers and $\sim$ 32 OR gates, or $\sim$ 100 GEs.

Hardware Cost of Remus-N2**.** Remus-N2 is similar to Remus-N1, with an additional mask V. Hence, the additional cost comes from the need to store and process this mask. The storage cost is simply 128 extra Flip-Flops. However, the processing cost can be tricky, especially since we adopt a serial concept for the implementation of ρ. Hence, we also adopt a serial concept for the processing of V. We assume that V will be updated in parallel to ρ and we merge the masking and ρ operations. Consequently, we need 64 XORs for the masking, 3 XORs for doubling, 5 XORs in order to correct the domain separation bits after each block (note that 3 bits are fixed), and 1 Flip-Flop for the serialization of doubling. Overall, we need $\sim$ 800 GEs on top of Remus-N1, $\sim 4,055$ GEs overall, which is again smaller than almost all other AEAD designs (except other Remus variants), while achieving BBB security.

Hardware Cost for Remus-N3**.** Remus-N3 uses Skinny-64/128 as the TBC (via the mode ICE3). According to our estimations, it requires 224 multiplexers, 32 XOR gates for the KDF function, 68 XORs for ρ, 24 XOR gates for correcting the Tweakey and 3 XOR gates for the counting. This adds up to $\sim$ 743 GEs on top of Skinny-64/128, which costs $1,399$ GEs. So we estimate that Remus-N3 costs $2,439$ GEs, which is a very small figure for any round-based implementation of an AEAD mode.

The arguments about serialization, unrolling and decryption are the same for all Remus-N variants. Thanks to the shared structure, these arguments also generally apply to Remus-M.

6.3 Primitives Choices

LFSR-Based Counters. The NIST call for lightweight AEAD algorithms requires that such algorithms must allow encrypting messages of length at least 2^{50} bytes while still maintaining their security claims. This means that using a TBC whose block size is 128 bits, we need a block counter of a period of at least 2^{46}. While this can be achieved by a simple arithmetic counter of 46 bits, arithmetic counters can be costly both in terms of area ($3 \sim 5$ GEs/bit) and performance (due to the long carry chains which limit the frequency of the circuit). In order to avoid this, we decided to use LFSR-based counters which can be implemented using a handful of XOR gates (3 XORs $\approx 6 \sim 9$ GEs). This, in addition to the architecture described above, makes the cost of counter almost negligible. For Remus, these counters are either dedicated counter, in the case of Remus-N3, or consecutive doubling of the key L, which is equivalent to a Galois LFSR. This, in addition to the architecture described above, makes the cost of counter almost negligible.

Counter Separation (Romulus-N2 and Romulus-M2). For Romulus-N2 and Romulus-M2, we used two LFSRs instead of one, such that the second LFSR is updated only when the first LFSR performs a full period. The rationale for that is to provide an even more lightweight variant of Romulus-N1 and Romulus-M1. The maximum input length imposed by NIST is quite large. In practice, the users may only need a support of input length up to $\sim 2^{28}$ bytes instead. In this case, the second LFSR does not need to be implemented. This saves $64 \sim 128$ Flip-Flops.

Smaller $\mathcal{D}$ for Romulus-N3. The goal of Romulus-N3 is to fit the Remus algorithm in Skinny-128-256, which is faster and smaller than Skinny-128-384. In most lightweight applications, the amount of data to be sent under the same key is small. Hence, Romulus-N3 represents a variant targeted at such applications that is faster and smaller than the other variants and can encrypt up to ~ 256 MBs of data.

Tag Generation. Considering hardware simplicity, the tag is the final output state (i.e., the same way as the ciphertext blocks), as opposed to the final state S of the TBC. In order to avoid branching when it comes to the output of the circuit, the tag is generated as $G(S)$ instead of S. In hardware, this can be implemented as $\rho(S, 0^n)$, i.e., similar to the encryption of a zero vector. Consequently, the output bus is always connected to the output of ρ and a multiplexer is avoided.

Padding. The padding function used in Remus is chosen so that the padding information is always inserted in the most significant byte of the last block of the message/AD. Hence, it reduces the number of decisions for each byte to only two decisions (either the input byte or a zero byte, except the most significant byte which is either the input byte or the byte length of that block). Besides, it is also the case when the input is treated as a string of words (16-, 32-, 64- or 128-bit words). This is much simpler than the classical 10* padding approach, where every word has a lot of different possibilities when it comes to the location of the padding string. Besides, usually implementations maintain the length of the message in a local variable/register, which means that the padding information is already available, just a matter of placing it in the right place in the message, as opposed to the decoder required to convert the message length into 10* padding.

Padding Circuit for Decryption. One of the main features of Remus is that it is inverse free and both the encryption and decryption algorithms are almost the same. However, it can be tricky to understand the behavior of decryption when the last ciphertext block has length $< n$. In order to understand padding in the decryption algorithm, we look at the ρ and ρ^{-1} functions when the input plaintext/ciphertext is partial. The ρ function applied on a partial plaintext block is shown in Equation (2). If ρ^{-1} is directly applied to $\mathtt{pad}_n(C)$, the corresponding output will be incorrect, due to the truncation of the last ciphertext block. Hence, before applying ρ^{-1} we need to regenerate the truncated bits. It can be verified that $C' = \mathtt{pad}_n(C) \oplus \mathtt{msb}_{n-|C|}(G(S))$. Once C' is regenerated, ρ^{-1} can be computed as shown in Equation (3):

$$\begin{bmatrix} S' \\ C' \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ G & 1 \end{bmatrix} \begin{bmatrix} S \\ \mathtt{pad}_n(M) \end{bmatrix} \quad \text{and} \quad C = \mathtt{lsb}_{|M|}(C'). \tag{2}$$

$$C' = \mathtt{pad}_n(C) \oplus \mathtt{msb}_{n-|C|}(G(S)) \quad \text{and} \quad \begin{bmatrix} S' \\ M \end{bmatrix} = \begin{bmatrix} 1 \oplus G & 1 \\ G & 1 \end{bmatrix} \begin{bmatrix} S \\ C' \end{bmatrix}. \tag{3}$$

While this looks like a special padding function, in practice it is simple. First of all, $G(S)$ needs to be calculated anyway. Besides, the whole operation can be implemented in two steps:

$$\begin{aligned} M &= C \oplus \mathtt{lsb}_{|C|}(G(s)), \\ S' &= \mathtt{pad}_n(M) \oplus S \end{aligned}$$

which can have a very simple hardware implementation, as discussed in the next paragraph.

Encryption-Decryption Combined Circuit. One of the goals of Romulus and Remus is to be efficient for implementations that require a combine encryption-decryption datapath. Hence, we made sure that the algorithm is inverse free, i.e., it does not used the inverse function of Skinny or $G(S)$. Moreover, ρ and ρ^{-1} can be implemented and combined using only one multiplexer, whose size depends on the size of the input/output bus. The same circuit can be used to solve the padding issue in decryption, by padding M instead of C. The tag verification operation simply checks if $\rho(S, 0^n)$ equals to T, which can be serialized depending on the implementation of ρ.

Choice of the G Matrix. We chose the position of G so that it is applied to the output state. This removes the need of G for AD processing, which improves software performance. In Section 3.1, we listed the security condition for G, and we choose our matrix G so that it meets these conditions and suits well for various hardware and software platforms.

We noticed that for lightweight applications, most implementations use an input/output bus of width $\leqslant 32$. Hence, we expect the implementation of ρ to be serialized depending on the bus size. Consequently, the matrix used in iCOFB can be inefficient as it needs a feedback operation over 4 bytes, which requires up to 32 extra Flip-Flops in order to be serialized, something we are trying to avoid in Remus. Moreover, the serial operation of ρ is different for byte, which requires additional multiplexers.

However, we observed that if the input block is interpreted in a different order, both problems can be avoided. First, it is impossible to satisfy the security requirements of G without any feedback signals, i.e., G is a bit permutation.

- If G is a bit permutation with at least one bit going to itself, then there is at least one non-zero value on the diagonal, so $I + G$ has at least 1 row that is all 0s.
- If G is a bit permutation without any bit going to itself, then every column in $I + G$ has exactly two 1's. The sum of all rows in such matrix is the 0 vector, which means the rows are linearly dependent. Hence, $I + G$ is not invertible.

However, the number of feedback signals can be adjusted to our requirements, starting from only 1 feedback signal. Second, we noticed that the input block/state of length n bits can be treated as several independent sub-blocks of size n/w each. Hence, it is enough to design a matrix G_s of size $w \times w$ bits and apply it independently n/w times to each sub-block. The operation applied on each sub-block in this case is the same (i.e., as we can distribute the feedback bits evenly across the input block). Unfortunately, the choice of w and G_s that provides the optimal results depends on the implementation architecture. However, we found out that the best trade-off/balance across different architectures is when $w = 8$ and G_s uses a single bit feedback.

In order to verify our observations, we generated a family of matrices with different values of w and G_s, and measured the cost of implementing each of them on different architectures.

7 Instantiation and Efficiency of Remus and Romulus

7.1 Instantiations of Remus and Romulus

We instantiated our two modes Remus and Romulus with the lightweight tweakable block cipher Skinny [BJK+16], which has the very interesting feature that if some part of the tweak is not used, the corresponding area for storage can be avoided. We tried to minimize the global control logic, the multiplexers, etc. More details about the various can be found in Appendix B.

7.2 Software efficiency

We refer to the Skinny document for discussions on software implementations of the various Skinny versions [BJK+16]. The Romulus mode will have little impact on the global performance of Skinny in software as long as serial implementations are used. We expect very little increase in ROM or RAM when compared to Skinny benchmarks. The very performant micro-controller implementations reported in the Skinny document were benchmarked without assuming parallel cipher calls, and without any pre-processing. Therefore, Romulus will present a very similar performance profile as the numbers reported on micro-controllers. Generally, using little amount of RAM, Skinny is easy and efficient to implement using a simple table-based approach.

For high-end platforms, such as latest Intel processors, very efficient highly-parallel bitsliced implementations of Skinny using SSE, AVX, AVX2 instructions on XMM/YMM registers will not be directly applicable as our Romulus mode is serial in nature. However, in a classical case of a server communicating with many lightweight devices, we note that it would be possible to consider bitslicing the key schedule [BGLP13] of Skinny (being relatively simple to compute) or using scheduling strategies [BLT15]. Classical table-based implementations of Skinny will ensure acceptable performance on even legacy platforms, while Vector Permute (`vperm`) might lead to better results on medium range platforms by parallelizing the computation of the Sbox.

7.3 ASIC Efficiency

We have implemented several variants of the Romulus and Remus families in the TSMC 65nm standard cell library, in order to have a better understanding of the ASIC performance and cost and verify our estimations in Section 6. The implementations use the round-based Skinny implementation published on the Skinny website [BJK+]. The results in Table 4 show that Remus is very lightweight and efficient at the same time as it requires slightly above 3 KGE (3.5 KGE with a simple interface) for its round-based implementation for Remus-N1. Of course, even smaller (but slower) trade-offs are possible by going with a serial implementation. At the other end of the spectrum in terms of security, the nonce-misuse resistant version Remus-M2 can be implemented in less than 5 KGE and provides 128-bit security, even in environments where randomness is not very reliable.

Table 4: ASIC Implementations of Remus and Romulus using the TSMC 65nm standard cell library.

Variant	**Cycles**	**Area** (GE)	**Minimum Delay** (ns)	**Throughput** (Gbps)	**Thput/Area** (Gbps/kGE)
Nonce-respecting schemes (64-bit data security)					
Remus-N1	44	3611	0.98	2.96	0.82
Remus-N1 Low Area	936	2834	0.8	0.1705	0.06
Nonce-respecting schemes (128-bit data security)					
Remus-N2 Low Area	936	3700	0.8	0.1705	0.046
Remus-N2	44	4774	0.84	3.46	0.72
Remus-N2 unrolled x4	14	9278	0.98	9.14	0.98
Romulus-N1 Low Area	1264	3390	0.75	0.195	0.06
Romulus-N1	60	6220	0.5	6.133	0.98
Romulus-N1 unrolled x2	32	7978	0.5	11.1	1.39
Romulus-N1 unrolled x4	18	10008	1	9.35	0.93
ACORN [Geo17]	-	6580	0.9	8.8	1.36
Ascon Low Area [Git17]	3078	4545	0.5	0.042	0.01
Ascon Basic Iterative [Git17]	6	8562	1	10.4	1.22
Nonce-misuse resistant schemes (64-bit data security)					
Remus-M1	44(AD)/88(M)	3805	1.01	2.16	0.56
Nonce-misuse resistant schemes (128-bit data security)					
Remus-M2	44(AD)/88(M)	4962	0.93	2.34	0.47

We have also implemented the serial, round-based and 4-round unrolled architectures of Romulus-N1. The figures are expected to be similar for Romulus-N2 and about 550 GEs smaller, and 12∼18% faster for Romulus-N3. Moreover, Remus and Romulus share a similar structure and our experimental results show that it is very cheap to convert the implementation to the misuse resistant variant. The implementation results of the Romulus-N1 and Remus-N1 4-round unrolled architecture show how Romulus and Remus take advantage of the versatility of Skinny for different architectures, where the throughput can be multiplied by 4 and the efficiency can be increased at a smaller area cost (about 75∼100%).

We have implemented Romulus-N1, Remus-N1 and Remus-N2 using the byte serial Skinny architecture and serialized feedback function for the modes. Serial implementations for different variants of Skinny have been proposed in [BJK+16, BJK+, JMPS17]. Such implementations can be easily adapted for other Romulus and Remus members. We have implemented one of our smallest proposals, Remus-N1 using the byte-serial implementation of Skinny. Without the interface, Remus-N1 requires around 2100 GEs using the TSMC 65nm standard cell library including the Skinny logic, Remus-N1 Logic and both the key and state storage.

We compare our implementations to the ones of Ascon and ACORN, the winners of the lightweight portfolio of the CAESAR competition. We have downloaded the implementation of ACORN from the ATHENa benchmarking website [Geo17] that were also used as part of the ASIC benchmarking efforts of the CAESAR competition [KHYKC17]. For Ascon, we have downloaded the serial and round based implementations from the designers' website [Git17], since these implementations seem to have better performance metrics compared to the one on the ATHENa website, at the expense of using a different interface. We have synthesized all four implementations to the same technology we are using, TSMC 65nm, to allow fair comparisons. We show that Romulus achieves comparable performance (but Romulus security proofs are in the standard model) while Remus is very competitive in terms of low area. For example, all the variants of Remus can achieve throughput above 1 Gbps for less than 5000 GEs, while the performance of Ascon drops significantly when the area is reduced to this range. Additionally, we believe the different security models and goals should be taken into consideration when comparing the performance, as in Romulus and Remus we use a much stronger primitive without any round reduction. However, we must note that Ascon is significantly better for energy efficiency using the round-based architecture because of this difference, as the round based implementation needs only 6 cycles per primitive call. A more thorough exploration of the design space of Romulus-N1 is provided in Section 8.

7.4 FPGA Efficiency

The FPGA results presented in Tables 5 show that the FPGA implementations of Romulus and Remus follow the same trend as the ASIC implementations achieving between 156 and 285 Slices for the round based implementations and less than 500 Slices for the 4-round unrolled implementations.

8 ASIC Design Space Exploration of Romulus-N1

In this section we provide a full list of different ASIC implementation trade-offs Romulus-N1, showing the design range of our proposals. First we study the impact of different number of round-unrolling, where `Rx` means round-based implementation with `x` round unrolling. We vary the operating frequency from the maximum possible frequency till 125 MHz and see the impact on throughput, area, power and energy. The results are given in Table 6.

Second, in Table 7, we study the byte serial architecture based on the bit-sliding technique, following the same methodology.

Finally, in Table 8, we study the 3-share threshold implementation of both the byte serial architecture (`PS`) and the single round architecture (`P1`). The implementations are based on the threshold implementations provided by the Skinny team [BJK+16].

The results show that the best throuput/area trade off is acheived by the `R2` architecture, which processes two rounds on the TBC per cycle. The minimum energy is acheived by 4-round unrolling (the `R4` architecture), as it requires only 18 cycles per block for

Table 5: FPGA Results for Remus and Romulus on the Xilinx Virtex 6 FPGA using ISE

Variant	Slices	LUTs	Registers	Max. Freq. (MHz)	Throughput (Mbps)	Throughput/Area (Mbps/Area)
Nonce-respecting schemes (64-bit data security)						
Remus-N1	189	540	308	250	727.7	3.8
Remus-N1 †	550	1669	338	147.7	1358.84	2.4
Nonce-respecting schemes (128-bit data security)						
Remus-N2	225	757	358	250	727.7	3.23
Romulus-N1	307	919	534	250	695	2.26
Romulus-N1 †	597	1884	528	250	2300	3.85
Lilliput-I-128	391	1506	1017	185	657.8	1.68
Lilliput-II-128	309	1088	885	185	328.9	1.06
Nonce-misuse resistant schemes (64-bit data security)						
Remus-M1	220	595	322	240	348	1.58
Nonce-misuse resistant schemes (128-bit data security)						
Remus-M2	279	816	397	219	317.6	1.13

† Unrolled x4.

Table 6: The design space of Romulus-N1 using the TSMC-65 ASIC technology: (unrolled) round-based implementations

Arch.	Critical Path (ns)	Area (GE)	Power (mW)	Energy (Enc block) (pJ)	Energy (Auth block) (pJ)	Throughput (Enc only) (Mbps)	Throughput (Auth only) (Mbps)	Throughput ($\vert A\vert = \vert M\vert$) (Mbps)	Thput/Area
R1	0.5	6220	0.8	24	12.8	4267	8000	6133	0.98
R1	0.66	5877	0.72	28.5	15.2	3232	6060	4647	0.79
R1	0.75	5864	0.65	29.25	15.6	2844	5333	4089	0.69
R1	1	5860	0.53	31.8	16.9	2133	4000	3067	0.52
R1	2	5860	0.35	42	22.4	1067	2000	1533	0.26
R1	4	5860	0.25	60	32	533	1000	767	0.13
R1	8	5772	0.22	105.6	56.3	266	500	383	0.07
R2	0.5	7978	0.99	15.84	8.91	8000	14200	11100	1.39
R2	0.66	7161	0.78	16.47	9.2	6060	10758	8410	1.17
R2	0.75	6860	0.71	17.04	9.5	5333	9467	7400	1.07
R2	1	6727	0.56	17.92	10.08	4000	7100	5550	0.82
R2	2	6635	0.38	24.32	13.68	2000	3550	2775	0.41
R2	4	6635	0.29	60	20.88	1000	1775	1388	0.2
R2	8	6635	0.25	64	36	500	888	694	0.1
R4	0.8	12766	1.1	15.84	9.68	8888	14500	11694	0.91
R4	1	10008	0.77	13.86	8.47	7111	11600	9356	0.93
R4	2	8740	0.46	16.56	10.12	3555	5800	4678	0.53
R4	4	8740	0.36	25.92	15.84	1775	2900	2339	0.26
R4	8	8740	0.32	46.08	28.16	889	1450	1170	0.13
R8	1.6	18679	1.2	21.12	14.4	7250	10662	8956	0.48
R8	2	15133	0.89	19.58	13.35	5800	8530	7165	0.47
R8	4	12990	0.51	22.44	15.3	2900	4265	3582	0.28
R8	8	12990	0.45	39.6	27	1450	2132	1791	0.14

encryption and 11 cycles per block for associated data. The serial implementation can be as low as 3.3 KGE, which is extremely low for a mode with full n-bit security and standard model security proofs. On the other hand, for about 8 KGE, we can have a performantthreshold implementation protected against side-channel attacks.

Table 7: The design space of Romulus-N1 using the TSMC-65 ASIC technology: byte serial implementations

Arch.	Critical Path (ns)	Area (GE)	Power (mW)	Energy (Enc block) (pJ)	Energy (Auth block) (pJ)	Throughput (Enc only) (Mbps)	Throughput (Auth only) (Mbps)	Throughput ($\|A\| = \|M\|$) (Mbps)	Thput/Area
S1	0.75	3390	0.5	489	247.5	131	259	195	0.06
S1	1	3318	0.5	652	330	98	194	146	0.04
S1	2	3318	0.29	756	383	49	97	73	0.02
S1	4	3318	0.2	1043	528	25	49	37	0.01
S1	8	3318	0.15	1565	792	12	24	18	0.005

Table 8: The design space of Romulus-N1 using the TSMC-65 ASIC technology: byte serial implementations and round based first-order threshold implementations

Arch.	Critical Path (ns)	Area (GE)	Power (mW)	Energy (Enc block) (pJ)	Energy (Auth block) (pJ)	Throughput (Enc only) (Mbps)	Throughput (Auth only) (Mbps)	Throughput ($\|A\| = \|M\|$) (Mbps)	Thput/Area
PS	0.75	5163	0.79	772	391	131	259	195	0.04
PS	1	5158	0.62	808	409	98	194	146	0.03
PS	2	5154	0.40	1043	529	49	97	73	0.01
PS	4	5154	0.27	1408	713	25	49	37	0.007
PS	8	5154	0.21	2190	1110	12	24	18	0.003
P1	0.5	8386	1.2	36	19.2	4267	8000	6133	0.73
P1	0.66	8101	0.89	35.2	18.7	3232	6060	4647	0.57
P1	0.75	8048	0.8	36	19.2	2844	5333	4089	0.51
P1	1	8048	0.69	41.4	22.1	2133	4000	3067	0.38
P1	2	8048	0.46	55.2	29.4	1067	2000	1533	0.19
P1	4	8048	0.35	84	44.8	533	1000	767	0.095
P1	8	8048	0.28	134.4	51.6	266	500	383	0.05

9 Conclusions

In this article, we presented Romulus and Remus, two new families of very lightweight and efficient authenticated encryption with associated data (AEAD) modes, providing provable security beyond the birthday bound. Our instantiations with Skinny lightweight tweakable block cipher are extremely efficient and have been submitted to the NIST lightweight competition [NIS19].

Acknowledgements

The authors would like to thank the anonymous referees and Christoph Dobraunig for their helpful comments and suggestions. The second and fourth authors are supported by Temasek Laboratories, Singapore.

References

[BBLT18] Subhadeep Banik, Andrey Bogdanov, Atul Luykx, and Elmar Tischhauser. SUNDAE: Small Universal Deterministic Authenticated Encryption for the Internet of Things. *IACR Trans. Symmetric Cryptol.*, 2018(3):1–35, 2018.

[BDJR97] Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. A Concrete Security Treatment of Symmetric Encryption. In *FOCS*, pages 394–403. IEEE Computer Society, 1997.

[BDPA11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. In *Selected Areas in Cryptography*, volume 7118 of *Lecture Notes in Computer Science*, pages 320–337. Springer, 2011.

[BGLP13] Ryad Benadjila, Jian Guo, Victor Lomné, and Thomas Peyrin. Implementing Lightweight Block Ciphers on x86 Architectures. In Tanja Lange, Kristin E. Lauter, and Petr Lisonek, editors, *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*, volume 8282 of *Lecture Notes in Computer Science*, pages 324–351. Springer, 2013.

[BGM04] Mihir Bellare, Oded Goldreich, and Anton Mityagin. The Power of Verification Queries in Message Authentication and Authenticated Encryption. *IACR Cryptology ePrint Archive*, 2004:309, 2004.

[BJK$^+$] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The Skinny Cipher Website.

[BJK$^+$16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In *CRYPTO (2)*, volume 9815 of *Lecture Notes in Computer Science*, pages 123–153. Springer, 2016.

[BLT15] Andrey Bogdanov, Martin M. Lauridsen, and Elmar Tischhauser. Comb to Pipeline: Fast Software Encryption Revisited. In Gregor Leander, editor, *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, volume 9054 of *Lecture Notes in Computer Science*, pages 150–171. Springer, 2015.

[BN08] Mihir Bellare and Chanathip Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. *J. Cryptology*, 21(4):469–491, 2008.

[BRW04] Mihir Bellare, Phillip Rogaway, and David A. Wagner. The EAX Mode of Operation. In *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 389–407. Springer, 2004.

[CAE] CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. See http://competitions.cr.yp.to/caesar.html.

[CDH$^+$12] Donghoon Chang, Morris Dworkin, Seokhie Hong, John Kelsey, and Mridul Nandi. A Keyed Sponge Construction with Pseudo-randomness in the Standard Model. NIST SHA-3 2012 Workshop, 2012.

[CDNY18] Avik Chakraborti, Nilanjan Datta, Mridul Nandi, and Kan Yasuda. Beetle Family of Lightweight and Secure Authenticated Encryption Ciphers. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 218–241, 2018.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The Random Oracle Methodology, Revisited (Preliminary Version). In *STOC*, pages 209–218. ACM, 1998.

[CIMN17a] Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, and Mridul Nandi. Blockcipher-Based Authenticated Encryption: How Small Can We Go? In

CHES, volume 10529 of *Lecture Notes in Computer Science*, pages 277–298. Springer, 2017.

[CIMN17b] Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, and Mridul Nandi. Blockcipher-based Authenticated Encryption: How Small Can We Go? (Full version of [CIMN17a]). *IACR Cryptology ePrint Archive*, 2017:649, 2017.

[CLS17] Benoît Cogliati, Jooyoung Lee, and Yannick Seurin. New Constructions of MACs from (Tweakable) Block Ciphers. *IACR Trans. Symmetric Cryptol.*, 2017(2):27–58, 2017.

[CS14] Shan Chen and John P. Steinberger. Tight Security Bounds for Key-Alternating Ciphers. In *EUROCRYPT*, volume 8441 of *Lecture Notes in Computer Science*, pages 327–350. Springer, 2014.

[DEMS14] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1. Submission to the CAESAR competition, 2014.

[DEMS16] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1. 2. *Submission to the CAESAR Competition*, 2016.

[DJN19] Nilanjan Datta, Ashwin Jha, and Mridul Nandi. REMUS [AD-INT]. NIST Lightweight Cryptography Project Forum: `https://groups.google.com/a/list.nist.gov/forum/#!forum/lwc-forum`, 2019.

[Geo17] George Mason University. ATHENa: Automated Tools for Hardware Evaluation. `https://cryptography.gmu.edu/athena/`, 2017.

[Git17] Github. ASCON-128 Hardware Design Document. `https://github.com/IAIK/ascon_hardware/blob/master/doc/ascon_hw_doc.pdf`, 2017.

[GWDE15] Hannes Groß, Erich Wenger, Christoph Dobraunig, and Christoph Ehrenhöfer. Suit up!–Made-to-Measure Hardware Implementations of ASCON. In *2015 Euromicro Conference on Digital System Design*, pages 645–652. IEEE, 2015.

[IIMP19] Akiko Inoue, Tetsu Iwata, Kazuhiko Minematsu, and Bertram Poettering. Cryptanalysis of OCB2: Attacks on Authenticity and Confidentiality. In *CRYPTO (1)*, volume 11692 of *Lecture Notes in Computer Science*, pages 3–31. Springer, 2019.

[IKM$^+$19] Tetsu Iwata, Mustafa Khairallah, Kazuhiko Minematsu, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Ling Sun. TGIF v1. Submission to NIST Lightweight Cryptography Project, 2019.

[IKMP19a] Tetsu Iwata, Mustafa Khairallah, Kazuhiko Minematsu, and Thomas Peyrin. Remus v1. Submission to NIST Lightweight Cryptography Project, 2019.

[IKMP19b] Tetsu Iwata, Mustafa Khairallah, Kazuhiko Minematsu, and Thomas Peyrin. Romulus v1. Submission to NIST Lightweight Cryptography Project, 2019.

[IMPS17] Tetsu Iwata, Kazuhiko Minematsu, Thomas Peyrin, and Yannick Seurin. ZMAC: A Fast Tweakable Block Cipher Mode for Highly Secure Message Authentication. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 34–65. Springer, 2017.

[JLM$^+$17] Ashwin Jha, Eik List, Kazuhiko Minematsu, Sweta Mishra, and Mridul Nandi. XHX - A Framework for Optimally Secure Tweakable Block Ciphers from

Classical Block Ciphers and Universal Hashing. LATINCRYPT 2017, 2017. available at https://eprint.iacr.org/2017/1075.

[JMPS17] Jérémy Jean, Amir Moradi, Thomas Peyrin, and Pascal Sasdrich. Bit-Sliding: A Generic Technique for Bit-Serial Implementations of SPN-based Primitives - Applications to AES, PRESENT and SKINNY. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 687–707. Springer, 2017.

[JNP14] Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. Tweaks and Keys for Block Ciphers: The TWEAKEY Framework. In *ASIACRYPT (2)*, volume 8874 of *Lecture Notes in Computer Science*, pages 274–288. Springer, 2014.

[KCP17] Mustafa Khairallah, Anupam Chattopadhyay, and Thomas Peyrin. Looting the LUTs: FPGA optimization of AES and AES-like ciphers for authenticated encryption. In *International Conference on Cryptology in India*, pages 282–301. Springer, 2017.

[KHYKC17] Sachin Kumar, Jawad Haj-Yihia, Mustafa Khairallah, and Anupam Chattopadhyay. A Comprehensive Performance Analysis of Hardware Implementations of CAESAR Candidates. *IACR Cryptology ePrint Archive*, 2017:1261, 2017.

[KR11] Ted Krovetz and Phillip Rogaway. The Software Performance of Authenticated-Encryption Modes. In *FSE*, volume 6733 of *Lecture Notes in Computer Science*, pages 306–327. Springer, 2011.

[LRW02] Moses Liskov, Ronald L. Rivest, and David A. Wagner. Tweakable Block Ciphers. In *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2002.

[Mej19] Alexandre Meje. REMUS [AD-INT]. NIST Lightweight Cryptography Project Forum: https://groups.google.com/a/list.nist.gov/forum/#!forum/lwc-forum, 2019.

[Men15] Bart Mennink. Optimally Secure Tweakable Blockciphers. In *FSE*, volume 9054 of *Lecture Notes in Computer Science*, pages 428–448. Springer, 2015.

[MI15] Kazuhiko Minematsu and Tetsu Iwata. Tweak-Length Extension for Tweakable Blockciphers. In *IMA Int. Conf.*, volume 9496 of *Lecture Notes in Computer Science*, pages 77–93. Springer, 2015.

[MMH$^+$14] Nicky Mouha, Bart Mennink, Anthony Van Herrewege, Dai Watanabe, Bart Preneel, and Ingrid Verbauwhede. Chaskey: An Efficient MAC Algorithm for 32-bit Microcontrollers. In *Selected Areas in Cryptography*, volume 8781 of *Lecture Notes in Computer Science*, pages 306–323. Springer, 2014.

[MRV15] Bart Mennink, Reza Reyhanitabar, and Damian Vizár. Security of Full-State Keyed Sponge and Duplex: Applications to Authenticated Encryption. In *ASIACRYPT (2)*, volume 9453 of *Lecture Notes in Computer Science*, pages 465–489. Springer, 2015.

[MV04] David A. McGrew and John Viega. The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In Anne Canteaut and Kapalee Viswanathan, editors, *Progress in Cryptology - INDOCRYPT 2004*, volume 3348 of *LNCS*, pages 343–355. Springer, 2004.

[NIS19] NIST. Lightweight Cryptography Project. https://csrc.nist.gov/Projects/Lightweight-Cryptography, 2019.

[NS20] Yusuke Naito and Takeshi Sugawara. Lightweight authenticated encryption mode of operation for tweakable block ciphers. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):66–94, 2020.

[Pat08] Jacques Patarin. The "Coefficients H" Technique. In *Selected Areas in Cryptography*, volume 5381 of *Lecture Notes in Computer Science*, pages 328–345. Springer, 2008.

[PS16] Thomas Peyrin and Yannick Seurin. Counter-in-Tweak: Authenticated Encryption Modes for Tweakable Block Ciphers. In *CRYPTO (1)*, volume 9814 of *Lecture Notes in Computer Science*, pages 33–63. Springer, 2016.

[RBB03] Phillip Rogaway, Mihir Bellare, and John Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Trans. Inf. Syst. Secur.*, 6(3):365–403, 2003.

[Rog04a] Phillip Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In *ASIACRYPT*, volume 3329 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2004.

[Rog04b] Phillip Rogaway. Nonce-Based Symmetric Encryption. In *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 2004.

[RS06] Phillip Rogaway and Thomas Shrimpton. A Provable-Security Treatment of the Key-Wrap Problem. In *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 373–390. Springer, 2006.

[WGZ$^+$16] Lei Wang, Jian Guo, Guoyan Zhang, Jingyuan Zhao, and Dawu Gu. How to Build Fully Secure Tweakable Blockciphers from Classical Blockciphers. In *ASIACRYPT (1)*, volume 10031 of *Lecture Notes in Computer Science*, pages 455–483, 2016.

[Wu14] Hongjun Wu. Acorn v1. Submission to the CAESAR competition, 2014.

A Security Proofs

Preliminaries. Throughout this section, we change the interface of ρ so that it works on partial input $X \in \{0,1\}^{\leqslant n}$ for the second argument by applying the padding and truncation internally: that is,

$$\begin{aligned}
&\rho(S, M) = (S', C) \text{ s.t. } C = M \oplus \mathtt{lmt}_{|M|}(G(S)), S' = S \oplus \mathtt{pad}_n(M) \\
&\rho^{-1}(S, C) = (S', M) \text{ s.t. } M = C \oplus \mathtt{lmt}_{|C|}(G(S)), \\
&S' = S \oplus \mathtt{pad}_n(M) = (\mathtt{lmt}_{|C|}(G(S)) \| 0 \dots) \oplus \mathtt{pad}_n(C) \oplus S.
\end{aligned}$$

We also define ρ_1 and ρ_2 as the component functions of ρ, i.e., $\rho(S, M) = (\rho_1(S, M), \rho_2(S, M))$. The inverse functions ρ_1^{-1} and ρ_2^{-1} are defined similarly.

One can see that this does not change the specification of Romulus and Remus if this ρ is directly applied to the last block, which we assume in the following analysis.

For positive integer i and j, let $(i)_j$ denote $i \cdot (i-1) \cdot \dots \cdot (i-(j-1))$.

A.1 Proofs of Romulus-N (Theorem 1)

We focus on the case that the underlying TBC inside Romulus-N is a TURP $\widetilde{\mathsf{P}} : \overline{\mathcal{T}} \times \mathcal{M} \to \mathcal{M}$ with $\overline{\mathcal{T}} = \mathcal{T} \times \mathcal{B} \times \mathcal{D}$, which we simply write Romulus-N. The derivation of computational analogues is standard [BDJR97]. What needs to be proved are

$$\mathbf{Adv}^{\mathtt{priv}}_{\mathsf{Romulus\text{-}N}}(\mathcal{A}) = 0, \tag{4}$$

$$\mathbf{Adv}^{\mathtt{auth}}_{\mathsf{Romulus\text{-}N}}(\mathcal{B}) \leqslant \frac{3q_d}{2^n} + \frac{2q_d}{2^\tau} \tag{5}$$

for (q_e, q_d)-privacy-adversary $\mathcal{A}$ and (q_e, q_d)-authenticity adversary $\mathcal{B}$. Both have no limitation in their computation time.

For $A \in \{0,1\}^*$, we say A has a AD blocks if it is parsed as $(A[1], \dots, A[a]) \overset{n,t}{\leftarrow} A$. Similarly for $M \in \{0,1\}^*$, we say M has m message blocks if $|M|_n = m$. For encryption query (N, A, M) or decryption query (N, A, C, T) of a AD blocks and m message blocks, we define

$$\tilde{a} = \lfloor a/2 \rfloor + 1.$$

We observe the total number of $\widetilde{\mathsf{P}}$ calls for encryption query (N, A, M) and decryption query (N, A, C, T) is $\tilde{a} + m$.

Let $(N_i, A_i, M_i) \in \mathcal{N} \times \mathcal{A} \times \mathcal{M}$ be the i-th encryption query. The corresponding response is $(C_i, T_i) \in \mathcal{M} \times \{0,1\}^\tau$, where $|C_i| = |M_i|$. The i-th verification query is written as (N'_i, A'_i, C'_i, T'_i) and the response is R_i which is M^*_i or $\perp$. Let $|M_i|_n = |C_i|_n = m_i$.
Proposition 1. *For any $S \in \{0,1\}^n$ and $M \in \{0,1\}^{\leqslant n}$, let $C = \rho_2(S, M)$, where we have $|M| = |C|$. Then for any $C' \in \{0,1\}^{\leqslant n}$ such that $C' \neq C$ and $w_{C'} = w_C$ (which equals w_M), we have*

$$\rho_1(S, M) \neq \rho_1^{-1}(S, C').$$

Proof. Let $M' = C' \oplus \mathtt{lmt}_{|C'|}(G(S))$. We have

$$\begin{aligned}
\rho_1(S, M) &= S \oplus \mathtt{pad}_n(M) = S \oplus \mathtt{pad}_n(C \oplus \mathtt{lmt}_{|C|}(G(S))) \\
\rho_1^{-1}(S, C') &= S \oplus \mathtt{pad}_n(M') = S \oplus \mathtt{pad}_n(C' \oplus \mathtt{lmt}_{|C'|}(G(S))), \text{ thus} \\
\rho_1(S, M) \oplus \rho_1^{-1}(S, C') &= \mathtt{pad}_n(C \oplus \mathtt{lmt}_{|C|}(G(S))) \oplus \mathtt{pad}_n(C' \oplus \mathtt{lmt}_{|C'|}(G(S)))
\end{aligned} \tag{6}$$

When $|M|(= |C|) = |C'|$, the last term of (6) is non-zero in the first $|C|$ bits. When $|M| \neq |C'|$, the padding function ensures the last padded byte has a difference, hence the last term of (6) is non-zero in the last byte. Note that the case $M = C = \varepsilon$ and $|C'| = n$ (so that $\mathtt{pad}_n(C) = \mathtt{pad}_n(C')$ by setting $C' = 0^n$) or the converse case ($C' = \varepsilon$ and $|C| = n$) has been excluded as we assumed $w_C = w'_C$. □

Proposition 2. *For $s \in \{0,1\}^n$, let S' be a uniformly distributed random variable over $\{0,1\}^n$ or $\{0,1\}^n \backslash \{s\}$. For any fixed $A, A', C, C' \in \{0,1\}^{\leqslant n}$, we have*

$$\Pr[\rho_1(S', A') = \rho_1(s, A)] \leqslant \frac{1}{2^n - 1},$$
$$\Pr[\rho_1^{-1}(S', C') = \rho_1^{-1}(s, C)] \leqslant \frac{1}{2^n - 1}.$$

Proof. The first inequality holds since $\rho_1(S', A') = S' \oplus \mathtt{pad}_n(A')$. For the second inequality, observe that

$$\begin{aligned}\rho_1^{-1}(S', C') &= S' \oplus \mathtt{pad}_n(C' \oplus \mathtt{lmt}_{|C'|}(G(S'))) \\ &= S' \oplus \mathtt{lmt}_{|C'|}(G(S')) \| 0^* \oplus \mathtt{pad}_n(C') \\ &= (G^{(|C'|)} + I) \cdot S' \oplus \mathtt{pad}_n(C').\end{aligned}$$

From the soundness of G, the last term of above equation has point probability at most $1/(2^n - 1)$. □

Proof of Privacy Bound (4)**.** In a similar manner to ΘCB3, proving (4) is immediate from the observation that each block in $\{C_1, \ldots, C_q, T_1, \ldots, T_q\}$ is generated from the output of $\widetilde{\mathsf{P}}$ given to G taking tweak $T \in \overline{\mathcal{T}}$ unique to each block, throughout the game. As G is sound (Definition 1), if Y is independent and random, so is $G(Y)$. The soundness of G also ensures the uniformity of the last ciphertext block $C[m]$ and the tag T.

Proof of Authenticity Bound (5)**.** We first prove the case $q_d = 1$ following the proof of Naito and Sugawara [NS20].

Let (N', A', C', T') denote the single verification query (here we omit the subscript), and let p be the probability of successful forgery. Let Θ_e be the random variable representing the transcripts obtained by encryption queries and responses. By convention, we can assume the authenticity adversary $\mathcal{B}$ as deterministic (since the optimal adversary is always deterministic). Moreover, we only need to consider the case that $\mathcal{B}$ performs q encryption queries first and then one verification query.

Let a_i and m_i be the number of AD and plaintext blocks in the i-th query. We write $X_i[j]$ and $Y_i[j]$ to denote the j-th TURP input and output in the encryption, for $j \in [\![m_i]\!]$. Here, $Y_i[j]$ is to encrypt $M_i[j+1]$ when $j < m_i$ and $X_i[m_i]$ is given to $\widetilde{\mathsf{P}}$ with tweak (N_i, w_{M_i}, m_i) to create $Y_i[m_i]$. Tag is $T_i = \mathtt{lmt}_\tau(\rho_2(Y_i[m_i], 0^n)) = \mathtt{lmt}_\tau(G(Y_i[m_i]))$.

By convention, let S_i be the state value after the AD process in the i-th encryption query, i.e. the output of $\widetilde{\mathsf{P}}$ with tweak (N_i, w_{A_i}, a_i) (see Figure 3). Note that $X_i[1] = \rho_1(S_i, M_i[1])$. Similarly, let a' and m' be AD and ciphertext block lengths of the single decryption query (N', A', C', T'), and define $X'[j]$, $Y'[j]$ and S' for the decryption query in the same fashion. For such $\mathcal{B}$, we have

$$\Pr[T' = T^*] \leqslant \max_{\theta_e \in \mathcal{S}_{\Theta_e}} \Pr[T' = T^* | \Theta_e = \theta_e],$$

where $\mathcal{S}_{\Theta_e}$ denotes the set of all possible values for θ_e, and T^* denotes the true tag value for (N', A', C', T'). We also observe that the distribution of Θ_e is a joint distribution of $(C^q, T^q, N^q, A^q, M^q)$, and C^q and T^q are uniformly distributed and independent from the values of (N^q, A^q, M^q) from (4) as they are the TURP outputs taking distinct tweak.

We fix θ_e and write $\Pr_e$ to denote the conditional probability space given $\Theta_e = \theta_e$. Also, let p_e denote the forgery probability given $\Theta_e = \theta_e$, i.e., $p_e = \Pr_e[T' = T^*] = \Pr[T' = T^* | \Theta_e = \theta_e]$ for the fixed θ_e.

All $X_i[j]$ and $Y_i[j]$ are determined by Θ_e except $j = m_i$: they are cut when the last block is partial or $\tau < n$. For simplicity, we let encryption oracle to disclose $X_i[m_i]$ and $Y_i[m_i]$ and attach to Θ_e *after* all the encryption queries.

Let T^* be the true tag value for decryption query, i.e.,

$$T^* = \mathtt{lmt}_\tau(G(Y'[m'])), \quad Y'[m'] = \widetilde{\mathsf{P}}^{(N', w_{C'}, m')}(X'[m']), \tag{7}$$

where $w_{C'}$ is the indicator of padding for C'. As we assumed G is regular, if $Y'[m']$ is uniformly distributed and independent from the previous variables (of transcript), the guessing probability of tag is $1/2^\tau$.

We do a case analysis.

Case 1: $N' \neq N_i$ for all i: since the final tweak of decryption query is $(N', w_{C'}, m')$ which has never been used before, from (7), $T^* = \mathtt{lmt}_\tau(G(Y'[m']))$ is unpredictable. $p_e = 1/2^\tau$.

Case 2: $N' = N_i$ for some $1 \leqslant i \leqslant q$. Wlog we let $N' = N_1$ and for simplicity omit the subscript i: (N, A, M, C, T) means $(N_1, A_1, M_1, C_1, T_1)$ and a and m mean a_1 and m_1. The same applies to $X_i[j]$ and $Y_i[j]$.

Case 2-1: $C' \neq C$, $(m, w_M) \neq (m', w_{C'})$. Then the final tweak values, (N, w_M, m) and $(N'(= N), w_{C'}, m')$, are different, and the latter never appeared in the encryption queries. Thus $Y'[m']$ is random from (7), and we have $p_e = 1/2^\tau$.

Case 2-2: $C' \neq C$, $(m, w_M) = (m', w_{C'})$. The block lengths of C and C' are the same, either both have partial last blocks or full blocks. Let $\delta \in [\![m]\!]$ be the block index of the *last* difference in C and C': $C[\delta] \neq C'[\delta]$ and $C[i] = C'[i]$ for all $\delta < i \leqslant m$. We have two important observations hold for any (δ, m): the first is

$$[X[m] = X'[m]] \Leftrightarrow [X[\delta] = X'[\delta]], \tag{8}$$

which follows from the fact that, we can write as $X[m] = \Phi(X[\delta])$ and $X'[m] = \Phi(X'[\delta])$ for some permutation Φ determined by $(C[\delta+1], \ldots, C[m])$ and N and $\widetilde{\mathsf{P}}$, thanks to the the soundness of G. The second one is

$$\begin{aligned} \Pr_e[T' = T^* | X[m] \neq X'[m]] &\leqslant \max_y \Pr_e[\mathtt{lmt}_\tau(G(Y'[m])) = y | X[m] \neq X'[m]] \quad (9) \\ &\leqslant \frac{2^{n-\tau}}{2^n - 1} \leqslant \frac{2}{2^\tau}, \end{aligned}$$

which holds since $X[m] \neq X'[m]$ implies $Y'[m]$ is uniform over $\{0,1\}^n \backslash \{Y[m]\}$ (note that all $X[i]$'s and $Y[i]$'s are fixed from $\Theta_e = \theta_e$). Thus we have

$$\begin{aligned} p_e &\leqslant \Pr_e[T' = T^* | X[m] \neq X'[m]] + \Pr_e[X[m] = X'[m]] \\ &\leqslant \frac{2}{2^\tau} + \Pr_e[X[m] = X'[m]] \\ &= \frac{2}{2^\tau} + \Pr_e[X[\delta] = X'[\delta]], \end{aligned} \tag{10}$$

where the last equality follows from (8). We further go into the four sub-cases, Case 2-2-1 to 2-2-4. We first describe the third and fourth cases for simplicity:

Case 2-2-3: $m > 1$ and $1 < \delta < m$. We have

$$\begin{aligned}
&\Pr_e[X[\delta] = X'[\delta]] \\
&\leqslant \Pr_e[X[\delta] = X'[\delta] | X[\delta-1] = X'[\delta-1]] + \Pr_e[X[\delta] = X'[\delta] | X[\delta-1] \neq X'[\delta-1]],
\end{aligned} \tag{11}$$

where the first term is 0. This is because $X[\delta-1] = X'[\delta-1]$ implies $Y[\delta-1] = Y'[\delta-1]$, which holds as the tweaks are identical, and $C[\delta] \neq C'[\delta]$ and the soundness of ρ imply $X[\delta] \neq X'[\delta]$. The second term is evaluated as

$$\begin{aligned}
&\Pr_e[X[\delta] = X'[\delta] | X[\delta-1] \neq X'[\delta-1]] \\
&\leqslant \Pr_e[\rho_1^{-1}(Y[\delta-1], C[\delta]) = \rho_1^{-1}(Y'[\delta-1], C'[\delta]) | X[\delta-1] \neq X'[\delta-1]] \\
&\leqslant \Pr_e[\rho_1^{-1}(Y[\delta-1], C[\delta]) = (G+I)(Y'[\delta-1]) \oplus C'[\delta] | X[\delta-1] \neq X'[\delta-1]] \\
&\leqslant \max_{y \in \{0,1\}^n} \Pr_e[(G+I)(Y'[\delta-1]) = y | X[\delta-1] \neq X'[\delta-1]] \\
&\leqslant \frac{1}{2^n - 1},
\end{aligned}$$

since $Y'[\delta-1]$ is uniform over $\{0,1\}^n \backslash \{Y[\delta-1]\}$ and the soundness of G (in particular, the regularity of $G + I$). Thus, the r.h.s. of (11) is at most $1/(2^n - 1) < 2/2^n$. Combined with (10), $p_e \leqslant 2/2^\tau + 2/2^n$.

Case 2-2-4: $m > 1$ and $\delta = m$. In a similar manner to Case 2-2-3, we have

$$\begin{aligned}
&\Pr_e[X[\delta] = X'[\delta]] \\
&\leqslant \Pr_e[X[\delta] = X'[\delta] | X[\delta-1] = X'[\delta-1]] + \Pr_e[X[\delta] = X'[\delta] | X[\delta-1] \neq X'[\delta-1]],
\end{aligned}$$

where the first term is 0. The second term is bounded similarly:

$$\begin{aligned}
&\Pr_e[X[\delta] = X'[\delta] | X[\delta-1] \neq X'[\delta-1]] \\
&\leqslant \Pr_e[\rho_1^{-1}(Y[\delta-1], C[\delta]) = \rho_1^{-1}(Y'[\delta-1], C'[\delta]) | X[\delta-1] \neq X'[\delta-1]] \\
&\leqslant \Pr_e[\rho_1^{-1}(Y[\delta-1], C[\delta]) = (G^{(|C'[\delta]|)} + I)(Y'[\delta-1]) \oplus C'[\delta] | X[\delta-1] \neq X'[\delta-1]] \\
&\leqslant \max_{y \in \{0,1\}^n} \Pr_e[(G^{(|C'[\delta]|)} + I)(Y'[\delta-1]) = y | X[\delta-1] \neq X'[\delta-1]] \\
&\leqslant \frac{1}{2^n - 1},
\end{aligned}$$

from Proposition 2.

Case 2-2-1: $m = 1$. We have $\delta = 1$. Let V be the input to $\widetilde{\mathsf{P}}$ that creates S for encryption query (N, A, M). Here, $S = \widetilde{\mathsf{P}}^{(N, w_A, a)}(V)$ and $X[1] = \rho_1(S, M[1]) = \rho_1^{-1}(S, C[1])$. Similarly we define V' and S' for decryption query (N', A', C', T'). First suppose $(w_A, a) = (w_{A'}, a')$. As we assumed $N = N'$, this means that both S and S' are created from $\widetilde{\mathsf{P}}^{(N, w_A, a)}$ (the same tweak). Then the analysis is the same as Case 2-2-3 by replacing $X[\delta-1]$ by V and $X'[\delta-1]$ by V'. Thus, $p_e \leqslant 2/2^\tau + 2/2^n$ for this case. Second, suppose $(w_A, a) \neq (w_{A'}, a')$. Then S' is uniformly random and independent of S, and thus $\Pr[X[1] = X'[1]] = 1/2^n$ and we have $p_e \leqslant 2/2^\tau + 1/2^n$ from (10). Taking the maximum, we have $p_e \leqslant 2/2^\tau + 2/2^n$.

Case 2-2-2: $m > 1$ and $\delta = 1$. The analysis is essentially the same as Case 2-2-1: we set V and V' instead of $X[\delta - 1]$ and $X'[\delta - 1]$, and derive $p_e \leqslant 2/2^\tau + 2/2^n$.

Case 3: $N' = N_i$ for some $i \in [\![q]\!]$, and $C' = C_i$ and (thus) $A' \neq A_i$. We define $X_i[j]$ and $Y_i[j]$ as the j-th TURP input and output in the AD process of i-th encryption query with a_i-block AD, for $j \in [\![\tilde{a}_i]\!]$ with $\tilde{a}_i \overset{\text{def}}{=} \lceil (a_i + 1)/2 \rceil$. For example, when $a_i = 3$ ($\tilde{a}_i = 2$) we have $(X_i[1], Y_i[1])$ and $(X_i[2], Y_i[2])$ where $S_i = Y_i[2]$, and when $a_i = 4$ ($\tilde{a}_i = 3$) we have $(X_i[1], Y_i[1])$, $(X_i[2], Y_i[2])$ and $(X_i[3], Y_i[3])$ where $S_i = Y_i[3]$. Note that $Y_i[j] = \widetilde{\mathsf{P}}^{A_i[2j],8,2j}(X_i[j])$ for even $j \leqslant a_i$. Similarly to Case 2, wlog we assume $i = 1$ and abbreviate A_i as A and so on. Note that $(m, w_C) = (m', w_{C'})$ holds. The analysis is similar to Case 2 but with some cares for differences in AD process and encryption. We observe

$$[X[m] = X'[m]] \Leftrightarrow [S = S'].$$

Also (9) holds as well. Thus

$$\begin{aligned} p_e &\leqslant \Pr_e[T' = T^* | X[m] \neq X'[m]] + \Pr_e[X[m] = X'[m]] \\ &\leqslant \frac{2}{2^\tau} + \Pr_e[X[m] = X'[m]] \\ &= \frac{2}{2^\tau} + \Pr_e[S = S']. \end{aligned} \tag{12}$$

Sub-cases are as follows.

Case 3-1: $(a, w_A) \neq (a', w_{A'})$. Then S' is random and independent of S as tweaks are different. Thus $\Pr_e[S = S'] = 1/2^n$. From (12), $p_e \leqslant 2/2^\tau + 1/2^n$.

Case 3-2: $(a, w_A) = (a', w_{A'})$. Let $\delta \in [\![a]\!]$ be the block index of the last difference in A and A': $A[\delta] \neq A'[\delta]$ and $A[i] = A'[i]$ for all $\delta < i \leqslant a$.

Case 3-2-1: $a = 1$. Then $\delta = 1$ and $A[a] \neq A'[a]$ which implies $S' \neq S$. From (12) we have $p_e \leqslant 2/2^\tau$.

Case 3-2-2: $a > 1$ and $\delta = 1$. The same as Case 3-2-1: we have $S \neq S'$ thus $p_e \leqslant 2/2^\tau$.

Case 3-2-3: $a > 1$ and $1 < \delta < a$ and δ is even. Then $Y'[\delta/2]$ is uniform and independent of $Y[\delta/2]$ because tweaks contain different elements ($A[\delta]$ and $A'[\delta]$). In the same manner to (8), we observe $[Y[\delta/2] = Y'[\delta/2]] \Leftrightarrow [S = S']$, therefore $\Pr_e[S = S'] = 1/2^n$ and $p_e \leqslant 2/2^\tau + 1/2^n$.

Case 3-2-4: $a > 1$ and $1 < \delta < a$ and δ is odd. The analysis is similar to Case 2-2-3. Let γ be $(\delta - 1)/2$. Here $X[\gamma + 1] = \rho_1(Y[\gamma], A[\delta])$. When $\gamma = 3$ (the minimum case) we slightly abuse the notation by assuming $Y[\gamma] = Y'[\gamma] = 0^n$. We have

$$\begin{aligned} \Pr_e[S = S'] \leqslant\ &\Pr_e[S = S' | X[\gamma] = X'[\gamma], A[\delta - 1] = A'[\delta - 1]] \\ &+ \Pr_e[S = S' | X[\gamma] = X'[\gamma], A[\delta - 1] \neq A'[\delta - 1]] \\ &+ \Pr_e[S = S' | X[\gamma] \neq X'[\gamma]]. \end{aligned} \tag{13}$$

For the first term of the right hand side, the conditional clause implies $Y[\gamma] = Y'[\gamma]$ and thus $X[\gamma + 1] \neq X'[\gamma + 1]$ from ρ taking different AD blocks ($A[\delta]$ and $A'[\delta]$)

and the property of G. This also implies $S \neq S'$, thus the probability is 0. For the second term, $Y'[\gamma]$ is uniform over $\{0,1\}^n$ (independent of $Y[\gamma]$) for different tweaks, thus $X[\gamma+1] = X'[\gamma+1]$ holds with probability $1/2^n$. Since $X[\gamma+1] = X'[\gamma+1]$ is equivalent to $S = S'$, the probability is $1/2^n$. For the third term, $Y'[\gamma]$ is either uniform over $\{0,1\}^n$ or $\{0,1\}^n \backslash \{Y[\gamma]\}$, thus $X[\gamma+1] = X'[\gamma+1]$ holds with probability at most $1/(2^n-1)$ thanks to the property of ρ. Therefore, (13) is at most $0 + 1/2^n + 1/(2^n-1) < 3/2^n$ and $p_e \leqslant 2/2^\tau + 3/2^n$.

Case 3-2-5: $a > 1$ and $\delta = a$. Similarly to Case 3-2-3, when $\delta = a$ is even, $Y'[\delta/2]$ is uniform over $\{0,1\}^n$ thus $X[\delta/2+1] = X'[\delta/2+1]$ (which is equivalent to $S = S'$) holds with probability $1/2^n$. Thus $p_e \leqslant 2/2^\tau + 1/2^n$. When $\delta = a$ is odd, the analysis of Case 3-2-4 applies. Here, $X[\gamma+1]$ is given to TURP taking nonce to produce S. We have $p_e \leqslant 2/2^\tau + 3/2^n$.

Therefore, for any fixed $\Theta_e = \theta_e$, $p_e \leqslant 3/2^n + 2/2^\tau$ holds for all cases. Thus the authenticity bound for $q_d = 1$ is $3/2^n + 2/2^\tau$. Using the generic conversion of the bound for $q_d = 1$ to $q_d \geqslant 1$ [BGM04], we conclude the proof of (5).

A.2 Security bounds of ICE in Remus

The internal TBC of Remus (ICE) is a variant of XHX [JLM+17], though the proof cannot be directly derived from that of XHX. We first prove the security of ICE, and then prove the security of Remus-N as a mode of ICE. Following [Men15, WGZ+16, JLM+17], the relevant security notion is defined as follows.

Definition 2. Let $\widetilde{E} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \to \mathcal{M}$ be a TBC of (key, tweak, message) space $(\mathcal{K}, \mathcal{T}, \mathcal{M})$. Assume $\widetilde{E}$ internally invokes an ideal cipher $E : \mathcal{K}' \times \mathcal{M} \to \mathcal{M}$ which is uniform over the set of all block ciphers, possibly multiple times for its encryption and decryption. Note that because E can be accessed freely, $\widetilde{E}$ must involve key $K \xleftarrow{\$} \mathcal{K}$ in its computation. Let $\widetilde{\mathsf{P}} : \mathcal{T} \times \mathcal{M} \to \mathcal{M}$ be the TURP. We define Tweakable Strong Pseudorandom Permutation (TSPRP)-advantage of $\widetilde{E}$ as

$$\mathbf{Adv}^{\mathtt{tsprp}}_{\widetilde{E}}(\mathcal{A}) \overset{\text{def}}{=} |\Pr[\mathcal{A}^{(\widetilde{E},\widetilde{E}^{-1}),(E,E^{-1})} \Rightarrow 1] - \Pr[\mathcal{A}^{(\widetilde{\mathsf{P}},\widetilde{\mathsf{P}}^{-1}),(E,E^{-1})} \Rightarrow 1]|,$$

where E is independent of $\widetilde{\mathsf{P}}$ and K inside $\widetilde{E}_K$.

This advantage shows the indistinguishability of $\widetilde{E}_K$ from TURP $\widetilde{\mathsf{P}}$, for adversaries accessing both encryption and decryption direction for both the construction ($\widetilde{E}_K$ in the real and $\widetilde{\mathsf{P}}$ in the ideal world) and the primitive (E in the both worlds). In this sense, a query to the former oracle will be called a construction query (c-query for short), and the latter will be called a primitive query (p-query for short). An adversary that uses q_c c-queries and q_p p-queries are called (q_c, q_p)-adversary, without a constraint on time complexity. By convention, if oracles are $(\widetilde{E}, \widetilde{E}^{-1})$ and (E, E^{-1}) (resp. $(\widetilde{\mathsf{P}}, \widetilde{\mathsf{P}}^{-1})$ and (E, E^{-1})), we say the adversary is in the real world (resp. the ideal world). We first show the TSPRP-advantage bounds of ICE1 and ICE2 and ICE3.

Security Bounds of ICE. We write $(T, M, C) \in \mathcal{T} \times \mathcal{M} \times \mathcal{M}$ (tweak, plaintext, ciphertext) to denote the tuple of query-response obtained from a construction query, and $(\widehat{K}, \widehat{X}, \widehat{Y}) \in \mathcal{K}' \times \mathcal{M} \times \mathcal{M}$ (key, plaintext, ciphertext) to denote that from a primitive query.

We have the following security bounds for ICE.

```
Algorithm $\mathsf{GICE1}_K(T, M)$
// $\nu \in \{0, n\}$
   1. $(N, D, B) \leftarrow T$
   2. $L \leftarrow E_K(N \,\|\, 0^{n-nl})$
   3. $V \leftarrow E_{K \oplus_{\|} 1}(L)$
   4. if $\nu = 0$ then $V \leftarrow 0^n$
   5. $H^1 \leftarrow 2^D V$
   6. $H^2 \leftarrow 2^D L \oplus_{\|} B$
   7. $X \leftarrow M \oplus H^1$
   8. $Y \leftarrow E_{H^2}(X)$
   9. $C \leftarrow Y \oplus H^1$
  10. return $C$
```

```
Algorithm $\mathsf{GICE2}_K(T, M)$
// $|K| = k$, $|N| = nl \leqslant k - 8$
   1. $(N, D, B) \leftarrow T$
   2. $L \leftarrow K \oplus (N \| 0^{k-nl})$
   3. $(L[1], L[2]) \xleftarrow{k-8,8} L$
   4. $H^2 \leftarrow (2^D L[1]) \,\|\, (L[2] \oplus B)$
   5. $C \leftarrow E_{H^2}(M)$
   6. return $C$
```

Figure 15: Abstraction of ICE1 and ICE2 (GICE1) and ICE3 (GICE2).

Lemma 1. *For any (q_c, q_p)-adversary $\mathcal{A}$,*

$$\mathbf{Adv}^{\mathsf{tsprp}}_{\mathsf{ICE1}}(\mathcal{A}) \leqslant \frac{9q_c^2 + 4q_cq_p}{2^n} + \frac{2q_p}{2^n},$$
$$\mathbf{Adv}^{\mathsf{tsprp}}_{\mathsf{ICE2}}(\mathcal{A}) \leqslant \frac{9q_c^2 + 4q_cq_p}{2^{2n}} + \frac{2q_p}{2^n},$$
$$\mathbf{Adv}^{\mathsf{tsprp}}_{\mathsf{ICE3}}(\mathcal{A}) \leqslant \frac{0.5q_c^2}{2^{k-8}} + \frac{q_c \cdot q_p}{2^k}.$$

Abstraction of ICE. We consider a TBC called GICE1 which generalizes ICE1 and ICE2 by introducing parameter $\nu \in \{0, n\}$ so that it derives L and V as ICE1 but $2^D V$ for $D \in \mathcal{D}$ is set to 0^n when $\nu = 0$, for any D. When $\nu = n$, $2^D V$ is computed over GF(2^n) and xored to the state. Figure 15 shows GICE1. We also abstract ICE1 and ICE2 w.r.t. the tweakey encoding and domain separation; the former is arbitrary injective mapping and the latter is a byte xored to the last byte of $2^D L$. For non-empty, possibly different length of binary strings, X and Y, let $X \oplus_{\|} Y$ denote the xor of X and Y, where the shorter one is prepended with zeros. Then $2^D L \oplus_{\|} B$ denotes $2^D L \oplus (0^{n-8} \| B)$. Here, $\mathcal{K}' = \mathcal{K} = \{0, 1\}^n$. An abstraction of ICE3 is presented as GICE2 in Figure 15. Note that the game allows decryption queries to GICE1 and GICE2, as well as bidirectional queries to E; they are straightforward and omitted in the figure. We have $|K| = k$ and $|N| = nl \leqslant k - 8$. Doubling is defined over GF(2^{k-8}). Note that in ICE3 we have $k = 128$, $nl = 96$.

Proofs of Lemma 1, for ICE1 and ICE2. We have the following lemma for GICE1.
Lemma 2. *For any (q_c, q_p)-adversary $\mathcal{A}$,*

$$\mathbf{Adv}^{\mathsf{tsprp}}_{\mathsf{GICE1}}(\mathcal{A}) \leqslant \frac{9q_c^2 + 4q_cq_p}{2^{n+\nu}} + \frac{2q_p}{2^n}.$$

Setting $\nu = 0$ gives the proof for ICE1, and $\nu = n$ gives the proof for ICE2.

In the following we prove Lemma 2. The proofs are based on Patarin's Coefficient-H technique [Pat08].

Since the game is purely information-theoretic, we can assume *wlog* that $\mathcal{A}$ is deterministic. We also assume that $\mathcal{A}$ never repeats queries.

Once a deterministic $\mathcal{A}$ is fixed, we can define the probability space of the transcript (the query-response pairs, which may include some additional variables to help analysis) defined as $\Theta = \theta$. We say θ is attainable if $\Pr_{\text{ideal}}[\Theta = \theta] > 0$, i.e. the probability under the ideal world is non-zero. Now we introduce the fundamental lemma of the Coefficient-H technique (see e.g. [Pat08, CS14] for details).

Lemma 3. *Assume that the set of all attainable transcripts is partitioned into the two disjoint sets,* GoodT *and* BadT*. If there exists* ϵ_1 *and* ϵ_2 *such that for any* $\theta \in \mathsf{GoodT}$*,*

$$\frac{\Pr_{\text{real}}[\Theta = \theta]}{\Pr_{\text{ideal}}[\Theta = \theta]} \geqslant 1 - \epsilon_1 \text{ and } \Pr_{\text{ideal}}[\Theta \in \mathsf{BadT}] \leqslant \epsilon_2,$$

then the advantage of $\mathcal{A}$ *is bounded by* $\epsilon_1 + \epsilon_2$*.*

Transcript in the Real World. We use the variables presented in Figure 15. Thus, for query-response tuple (T, M, C), we have $H^1 = 2^D V$, and $H^2 = 2^D L \oplus_{\shortparallel} B$, and $X = M \oplus H^1$, and $Y = C \oplus H^1$, where $Y = E_{H^2}(X)$. The *transcript* is defined as $\Theta = (\Theta_c, \Theta_p, \Theta_a)$, where $\Theta_c = (T_i(= (N_i, D_i, B_i)), M_i, C_i)_{i \in [\![q_c]\!]}$, $\Theta_p = (\widehat{K}_i, \widehat{X}_i, \widehat{Y}_i)_{i \in [\![q_p]\!]}$, and $\Theta_a = (L_i, V_i)_{i \in [\![q_c]\!]} \cup \{K\}$. We have some remarks. First, Θ_a is hidden information and is attached to Θ *after* all (construction and primitive) queries are made, but before making the final decision. This allows to determine all input/output of E invoked in the game from Θ, which simplifies the analysis. A similar trick has been commonly used in many Coefficient-H-based proofs. Note that Θ does not contain the directions of queries (encryption or decryption) as such information is irrelevant. Note that N_i may repeat from the definition of the game. Let c_N be the number of unique N_i's in c-queries.

Without loss of generality, the adversary is deterministic and has no duplicate queries, no redundant queries, e.g., a pair of encryption query $(T, M) \to C$ and decryption query $(T, C) \to M$. For random variable in a transcript, we may write the assignment by the corresponding lower case letter, e.g. $K = k$ and $H_i^2 = h_i^2$ etc.

Transcript in the Ideal World. Using two additional (secret) n-bit URPs, P_1 and P_2, we have $K \xleftarrow{\$} \mathcal{K}$, $L_i \leftarrow \mathsf{P}_1(N_i)$, and $V_i \leftarrow \mathsf{P}_2(N_i)$. Moreover, a TURP $\widetilde{\mathsf{P}}$ encrypts as $C_i \leftarrow \widetilde{\mathsf{P}}(T_i, M_i)$ when i-th c-query is an encryption query or $M_i \leftarrow \widetilde{\mathsf{P}}^{-1}(T_i, C_i)$ when i-th c-query is a decryption query.

Bad Event Definitions. Let $\mathcal{IN}_j$ be a set of (Key, Plaintext) input to E, where $j \in [\![4]\!]$ denotes the type of input:

$$\begin{aligned}
\mathcal{IN}_1 &= (K, N_i)_{i \in [\![q_c]\!]}, \\
\mathcal{IN}_2 &= (K \oplus_{\shortparallel} 1, L_i)_{i \in [\![q_c]\!]}, \\
\mathcal{IN}_3 &= (H_i^2, X_i)_{i \in [\![q_c]\!]}, \\
\mathcal{IN}_4 &= (\widehat{K}_i, \widehat{X}_i)_{i \in [\![q_p]\!]}.
\end{aligned}$$

Similarly, let $\mathcal{OU}_j$ for $j \in [\![4]\!]$ be a set of (Key, Ciphertext) output of E:

$$\begin{aligned}
\mathcal{OU}_1 &= (K, L_i)_{i \in [\![q_c]\!]}, \\
\mathcal{OU}_2 &= (K \oplus_{\shortparallel} 1, V_i)_{i \in [\![q_c]\!]}, \\
\mathcal{OU}_3 &= (H_i^2, Y_i)_{i \in [\![q_c]\!]}, \\
\mathcal{OU}_4 &= (\widehat{K}_i, \widehat{Y}_i)_{i \in [\![q_p]\!]}.
\end{aligned}$$

For $(i,j) \in [\![4]\!]^2$, let $\mathbf{BI}_{i,j}$ be a non-trivial collision event in the elements of $\mathcal{IN}_i$ and $\mathcal{IN}_j$, except $\mathbf{BI}_{1,4}$ and $\mathbf{BI}_{2,4}$, which are defined as key-input collision for the sake of simplicity. In a similar manner, we define $\mathbf{BO}_{i,j}$ for $(i,j) \in [\![4]\!]^2$ as a non-trivial collision event in the elements of $\mathcal{OU}_i$ and $\mathcal{OU}_j$, See below for the exact definitions. Let (the bad event) `BadT` be the set of transcripts satisfying one of $\mathbf{BI}_{i,j}$ or $\mathbf{BO}_{i,j}$.

Core Idea of the Proof. The analysis below is basically a variant of the proof of XHX which can be seen as an extension of the proof of XTX [MI15]. The (so-called) bad events are all $\mathbf{BI}_{i,j}$ and $\mathbf{BO}_{i,j}$ for $(i,j) \in [\![4]\!]^2$ and the occurrence of any of bad event implies a win of the adversary. Note that some cannot be possible by definition (see below). For example, when $\mathbf{BI}_{3,3}$ occurs for i-th and j-th c-queries (directions of queries do not matter) in the real world, we have $Y_i = Y_j$ equivalent to $H_i^1 \oplus C_i = H_j^1 \oplus C_j$ with probability one, which is observable in the transcripts (as H^1 and H^2 are computable). However, when $\mathbf{BI}_{3,3}$ occurs in the ideal world, $H_i^1 \oplus C_i = H_j^1 \oplus C_j$ occurs with a negligible probability, implying a distinguisher. Similar arguments apply to $\mathbf{BO}_{3,3}$. To evaluate the separation between KDF and main encryption, and the separation between input-output tuples of E created from p-queries and c-queries, we need more bad events.

We describe the definitions of $\mathbf{BI}_{i,j}$ and $\mathbf{BO}_{i,j}$ and their possibilities, the probability of event in the ideal world. For completeness, we list all the events including those not possible in the both worlds. All probabilities in this case analysis is about the ideal world.

- $\mathbf{BI}_{1,1} = [^\exists(i,j) \in [\![q_c]\!]^2, i \neq j : (K, N_i) = (K, N_j), N_i \neq N_j]$
  This is not possible for the both worlds.

- $\mathbf{BI}_{1,2} = [^\exists(i,j) \in [\![q_c]\!]^2 : (K, N_i) = (K \oplus_{\shortparallel} 1, L_j)]$
  This is not possible for the both worlds.

- $\mathbf{BI}_{2,2} = [^\exists(i,j) \in [\![q_c]\!]^2, i \neq j : (K \oplus_{\shortparallel} 1, L_i) = (K \oplus_{\shortparallel} 1, L_j), N_i \neq N_j]$
  This is not possible for both worlds (for the ideal world, when $N_i \neq N_j$ then $L_i \neq L_j$).

- $\mathbf{BI}_{1,3} = [^\exists(i,j) \in [\![q_c]\!]^2 : (K, N_i) = (H_j^2, X_j)]$
  This is possible for the both worlds. The maximum pairwise (i.e., between two queries) probability for the ideal world is

$$\begin{aligned}
&\Pr[K = 2^{D_j} L_j \oplus_{\shortparallel} B_j] \cdot \Pr[N_i = M_j \oplus 2^{D_j} V_j | K = 2^{D_j} L_j \oplus_{\shortparallel} B_j] \\
&\leqslant \max_c \Pr[K = c] \cdot \max_{d,e} \Pr[V_j = d | L_j = e] \\
&= \frac{1}{2^n} \cdot \frac{1}{2^\nu}.
\end{aligned}$$

  We remark that when $\nu = 0$ we have $V_j = 0^n$ thus the above holds true.
  The total (i.e., $\forall(i,j) \in [\![q_c]\!]^2$) probability is at most $c_N \cdot q_c / 2^{n+\nu} \leqslant q_c^2 / 2^{n+\nu}$.

- $\mathbf{BI}_{2,3} = [^\exists(i,j) \in [\![q_c]\!]^2 : (K \oplus_{\shortparallel} 1, L_i) = (H_j^2, X_j)]$
  This is possible for both worlds. The maximum pairwise probability is

$$\begin{aligned}
&\Pr[K \oplus_{\shortparallel} 1 = 2^{D_j} L_j \oplus_{\shortparallel} B_j] \cdot \Pr[L_i = M_j \oplus 2^{D_j} V_j | K \oplus_{\shortparallel} 1 = 2^{D_j} L_j \oplus_{\shortparallel} B_j] \\
&\leqslant \max_c \Pr[K = c] \cdot \max_{d,e} \Pr[V_j = d | L_j = e] \\
&= \frac{1}{2^n} \cdot \frac{1}{2^\nu}.
\end{aligned}$$

  The total probability is at most $q_c^2 / 2^{n+\nu}$.

- $\mathbf{BI}_{3,3} = [\exists(i,j) \in [\![q_c]\!]^2, i \neq j : (H_i^2, X_i) = (H_j^2, X_j)]$
  This is possible for both worlds. The maximum pairwise probability is:

$$\begin{aligned}&\Pr[2^{D_i} L_i \oplus_{\shortparallel} B_i = 2^{D_j} L_j \oplus_{\shortparallel} B_j] \\ &\quad \cdot \Pr[M_i \oplus 2^{D_i} V_i = M_j \oplus 2^{D_j} V_j | 2^{D_i} L_i \oplus_{\shortparallel} B_i = 2^{D_j} L_j \oplus_{\shortparallel} B_j]\end{aligned}$$

  This is zero when $T_i = T_j$ ($H_i^1 = H_j^1$, $H_i^2 = H_j^2$, $M_i \neq M_j$), or when $T_i \neq T_j$, $N_i = N_j$, $D_i = D_j$, $B_i \neq B_j$. When $T_i \neq T_j$ and $N_i \neq N_j$, it is at most

$$\max_c \Pr[L_i \oplus L_j = c] \cdot \max_{d,e} \Pr[V_i \oplus V_j = d | L_i \oplus L_j = e] \leqslant \frac{1}{2^n - 1} \frac{2^{n-\nu}}{2^n - 1}.$$

  When $T_i \neq T_j$, $N_i = N_j$, $D_i \neq D_j$, it is at most

$$\max_c \Pr[L_i = c] \cdot \max_{d,e} \Pr[V_i = d | L_i = e] \leqslant \frac{1}{2^n} \frac{1}{2^\nu}.$$

  Thus the pairwise probability is at most $\frac{2}{2^n} \cdot \frac{2}{2^\nu} \leqslant \frac{4}{2^{n+\nu}}$.
  The total probability is at most $4\binom{q_c}{2}/2^{n+\nu} \leqslant 2q_c^2/2^{n+\nu}$.

- $\mathbf{BI}_{1,4} = [\exists j \in [\![q_p]\!] : K = \widehat{K}_j]$
  This is possible for both worlds. The maximum pairwise probability is $1/2^n$, and the total probability is at most $q_p/2^n$.

- $\mathbf{BI}_{2,4} = [\exists j \in [\![q_p]\!] : K \oplus_{\shortparallel} 1 = \widehat{K}_j]$
  This is possible for the both worlds. The maximum pairwise probability is $1/2^n$, and the total probability is at most $q_p/2^n$.

- $\mathbf{BI}_{3,4} = [\exists(i,j) \in [\![q_c]\!] \times [\![q_p]\!] : (H_i^2, X_i) = (\widehat{K}_j, \widehat{X}_j)]$
  This is possible for the both worlds. The maximum pairwise probability is

$$\Pr[2^{D_i} L_i \oplus_{\shortparallel} B_i = \widehat{K}_j] \cdot \Pr[M_i \oplus 2^{D_i} V_i = \widehat{X}_j | 2^{D_i} L_i \oplus_{\shortparallel} B_i = \widehat{K}_j] \leqslant \frac{1}{2^n} \cdot \frac{1}{2^\nu}.$$

  The total probability is at most $q_c q_p / 2^{n+\nu}$.

- $\mathbf{BI}_{4,4} = [\exists(i,j) \in [\![q_p]\!]^2, i \neq j : (\widehat{K}_i, \widehat{X}_i) = (\widehat{K}_j, \widehat{X}_j)]$
  This is not possible for the both worlds.

- $\mathbf{BO}_{1,1} = [\exists(i,j) \in [\![q_c]\!]^2, i \neq j : (K, L_i) = (K, L_j), N_i \neq N_j]$
  This is not possible for the both worlds.

- $\mathbf{BO}_{1,2} = [\exists(i,j) \in [\![q_c]\!]^2 : (K, L_i) = (K \oplus_{\shortparallel} 1, V_j)]$
  This is not possible for the both worlds.

- $\mathbf{BO}_{2,2} = [\exists(i,j) \in [\![q_c]\!]^2, i \neq j : (K \oplus_{\shortparallel} 1, V_i) = (K \oplus_{\shortparallel} 1, V_j), N_i \neq N_j]$
  This is not possible for both worlds: for the ideal world, when $N_i \neq N_j$ then $V_i \neq V_j$.

- $\mathbf{BO}_{1,3} = [\exists(i,j) \in [\![q_c]\!]^2 : (K, L_i) = (H_j^2, Y_j)]$
  This is possible for both worlds. The maximum pairwise probability is

$$\begin{aligned}&\Pr[K = 2^{D_j} L_j \oplus_{\shortparallel} B_j] \cdot \Pr[L_i = C_j \oplus 2^{D_j} V_j | K = 2^{D_j} L_j \oplus_{\shortparallel} B_j] \\ &\leqslant \max_c \Pr[K = c] \cdot \max_{d,e} \Pr[V_j = d | L_j = e] \\ &= \frac{1}{2^n} \cdot \frac{1}{2^\nu}.\end{aligned}$$

  The total probability is at most $q_c^2 / 2^{n+\nu}$.

- $\mathbf{BO}_{2,3} = [\exists(i,j) \in [\![q_c]\!]^2 : (K \oplus_{\shortparallel} 1, V_i) = (H_j^2, Y_j)]$
  This is possible for both worlds. The maximum pairwise probability is

$$\begin{aligned}&\Pr[K \oplus_{\shortparallel} 1 = 2^{D_j} L_j \oplus_{\shortparallel} B_j] \cdot \Pr[V_i = C_j \oplus 2^{D_j} V_j | K \oplus_{\shortparallel} 1 = 2^{D_j} L_j \oplus_{\shortparallel} B_j] \\ &\leqslant \max_c \Pr[K = c] \cdot \Pr[V_i \oplus 2^{D_j} V_j = C_j | K \oplus_{\shortparallel} 1 = 2^{D_j} L_j \oplus_{\shortparallel} B_j] \\ &\leqslant \frac{1}{2^n} \cdot \Pr[V_i \oplus 2^{D_j} V_j = C_j | K \oplus_{\shortparallel} 1 = 2^{D_j} L_j \oplus_{\shortparallel} B_j].\end{aligned}$$

  This is at most $1/2^n$ when $\nu = 0$. When $\nu = n$, it is at most

$$\begin{aligned}&\frac{1}{2^n} \cdot \begin{cases} \max_e \Pr[(2^{D_j}+1)V_i = C_j, |L_j = e] \leqslant \frac{1}{2^n} & \text{if } N_i = N_j \\ \max_e \Pr[V_i = 2^{D_j} V_j \oplus C_j | L_j = e] \leqslant \frac{1}{2^n - 1} & \text{if } N_i \neq N_j \end{cases} \\ &\leqslant \frac{1}{2^n} \cdot \frac{1}{2^n - 1} \leqslant \frac{2}{2^{2n}},\end{aligned}$$

  where $N_i = N_j$ includes the case $i = j$. The total probability is thus bounded by $2q_c^2/2^{n+\nu}$.

- $\mathbf{BO}_{3,3} = [\exists(i,j) \in [\![q_c]\!]^2, i \neq j : (H_i^2, Y_i) = (H_j^2, Y_j)]$
  This is possible for the both worlds. The maximum pairwise probability is

$$\begin{aligned}&\Pr[2^{D_i} L_i \oplus_{\shortparallel} B_i = 2^{D_j} L_j \oplus_{\shortparallel} B_j] \\ &\quad \cdot \Pr[C_i \oplus 2^{D_i} V_i = C_j \oplus 2^{D_j} V_j | 2^{D_i} L_i \oplus_{\shortparallel} B_i = 2^{D_j} L_j \oplus_{\shortparallel} B_j].\end{aligned}$$

  With the same analysis as $\mathbf{BI}_{3,3}$, this is bounded by $4/2^{n+\nu}$. Thus, the total probability is at most $4\binom{q_c}{2}/2^{n+\nu} \leqslant 2q_c^2/2^{n+\nu}$.

- $\mathbf{BO}_{1,4} = [\exists(i,j) \in [\![q_c]\!] \times [\![q_p]\!] : (K, L_i) = (\widehat{K}_j, \widehat{Y}_j)]$
  This is possible for both worlds. The maximum pairwise probability is $1/2^{2n}$.
  The total probability is at most $q_c q_p / 2^{2n}$.

- $\mathbf{BO}_{2,4} = [\exists(i,j) \in [\![q_c]\!] \times [\![q_p]\!] : (K \oplus_{\shortparallel} 1, V_i) = (\widehat{K}_j, \widehat{Y}_j)]$
  This is possible for the both worlds. The maximum pairwise probability is $1/2^n \cdot 1/2^n$.
  The total probability is at most $q_c \cdot q_p / 2^{2n}$

- $\mathbf{BO}_{3,4} = [\exists(i,j) \in [\![q_c]\!] \times [\![q_p]\!] : (H_i^2, Y_i) = (\widehat{K}_j, \widehat{Y}_j)]$
  This is possible for both worlds. The maximum pairwise probability is

$$\Pr[2^{D_i} L_i \oplus_{\shortparallel} B_i = \widehat{K}_j] \cdot \Pr[C_i \oplus 2^{D_i} V_i = \widehat{Y}_j | 2^{D_i} L_i \oplus_{\shortparallel} B_i = \widehat{K}_j] \leqslant \frac{1}{2^n} \cdot \frac{1}{2^\nu}.$$

  The total probability is at most $q_c q_p / 2^{n+\nu}$.

- $\mathbf{BO}_{4,4} = [\exists(i,j) \in [\![q_p]\!]^2, i \neq j : (\widehat{K}_i, \widehat{Y}_i) = (\widehat{K}_j, \widehat{Y}_j)]$
  This is not possible for the both worlds.

Sum of Bad Event Probabilities. Bad event probability is the sum of all sub-bad-event probabilities:

$$\begin{aligned}
\Pr_{\text{ideal}}[\Theta \in \texttt{BadT}] &= \underbrace{q_c^2/2^{n+\nu}}_{\mathbf{BI}_{1,3}} + \underbrace{q_c^2/2^{n+\nu}}_{\mathbf{BI}_{2,3}} + \underbrace{2q_c^2/2^{n+\nu}}_{\mathbf{BI}_{3,3}} + \underbrace{q_p/2^n}_{\mathbf{BI}_{1,4}} + \underbrace{q_p/2^n}_{\mathbf{BI}_{2,4}} + \underbrace{q_cq_p/2^{n+\nu}}_{\mathbf{BI}_{3,4}} \\
&+ \underbrace{q_c^2/2^{n+\nu}}_{\mathbf{BO}_{1,3}} + \underbrace{2q_c^2/2^{n+\nu}}_{\mathbf{BO}_{2,3}} + \underbrace{2q_c^2/2^{n+\nu}}_{\mathbf{BO}_{3,3}} + \underbrace{q_cq_p/2^{2n}}_{\mathbf{BO}_{1,4}} + \underbrace{q_cq_p/2^{2n}}_{\mathbf{BO}_{2,4}} + \underbrace{q_cq_p/2^{n+\nu}}_{\mathbf{BO}_{3,4}} \\
&= \frac{8q_c^2 + 2q_cq_p}{2^{n+\nu}} + \frac{2q_p}{2^n} + \frac{q_c^2 + 2q_cq_p}{2^{2n}} \\
&\leqslant \frac{9q_c^2 + 4q_cq_p}{2^{n+\nu}} + \frac{2q_p}{2^n}. \qquad (14)
\end{aligned}$$

Good Event Probability Ratio. We introduce some notations. For the multiset $\mathcal{S}$, let $\mathsf{uni}(\mathcal{S})$ be the set of all unique elements of $\mathcal{S}$. Let

- $\mathcal{T}^\sharp = \{T_1^\sharp, \ldots, T_\alpha^\sharp\} = \mathsf{uni}(\mathcal{T})$ for $\mathcal{T} = \{T_1, \ldots, T_{q_c}\}$. Let $t(j)$ be the multiplicity of $T_j^\sharp$ in $\mathcal{T}$. We have $\sum_{j=1}^{\alpha} t(j) = q_c$.
- $\widehat{\mathcal{K}}^\sharp = \{\widehat{K}_1^\sharp, \ldots, \widehat{K}_\beta^\sharp\} = \mathsf{uni}(\widehat{\mathcal{K}})$ for $\widehat{\mathcal{K}} = \{\widehat{K}_1, \ldots, \widehat{K}_{q_p}\}$. Let $\widehat{k}(j)$ be the multiplicity of $\widehat{K}_j^\sharp$ in $\widehat{\mathcal{K}}$. We have $\sum_{j=1}^{\beta} \widehat{k}(j) = q_p$.
- $\mathcal{H}^{2\sharp} = \{H_1^{2\sharp}, \ldots, H_\gamma^{2\sharp}\} = \mathsf{uni}(\mathcal{H}^2)$ for $\mathcal{H}^2 = \{H_1^2, \ldots, H_{q_c}^2\}$. Let $h^{2\sharp}(j)$ be the multiplicity of $h_j^{2\sharp}$ in $\mathcal{H}^2$. We have $\sum_{j=1}^{\gamma} h^{2\sharp}(j) = q_c$.

Recall that K and $K \oplus_{\shortparallel} 1$ and all elements of $\widehat{K}^\sharp$ are distinct for any good transcript. We fix a good transcript $(\Theta_c = \theta_c, \Theta_p = \theta_p, \Theta_a = \theta_a)$, which determines the assignments of all internal variables, e.g., as $K = k$ or $H_1^2 = h_1^2$. For $i \in [\![\gamma]\!]$, let

$$\Sigma(i) = \begin{cases} 2^n - \widehat{k}(j) & \text{if there exists } j \in [\![\gamma]\!] \text{ s.t. } h_i^{2\sharp} = \widehat{k}_j^\sharp \\ 2^n - c_N & \text{if } h_i^{2\sharp} \in \{k, k \oplus_{\shortparallel} 1\} \\ 2^n & \text{otherwise.} \end{cases}$$

Note that for the first case, $h_i^{2\sharp} \notin \{k, k \oplus_{\shortparallel} 1\}$ holds from the absence of $\mathbf{BI}_{1,4}$ and $\mathbf{BI}_{2,4}$, and for the second case, KDF invokes keys k and $k \oplus_{\shortparallel} 1$ c_N times.

Since θ_p and θ_a determine $2^n - \Sigma(i)$ input/output pairs of $E_{h_i^{2\sharp}}(*)$ for $i \in [\![\gamma]\!]$, $\Sigma(i)$ denotes the number of inputs and outputs of random permutation $E_{h_i^{2\sharp}}(*)$ still undetermined with $\Theta_p = \theta_p$ and $\Theta_a = \theta_a$.

Transcript Probability in the Ideal World. For any attainable $\theta = (\theta_c, \theta_p, \theta_a) \notin \texttt{BadT}$, we have

$$\Pr_{\text{ideal}}[\Theta = \theta] = \Pr_{\text{ideal}}[\Theta_c = \theta_c | \Theta_p = \theta_p, \Theta_a = \theta_a] \cdot \Pr_{\text{ideal}}[\Theta_p = \theta_p | \Theta_a = \theta_a] \cdot \Pr_{\text{ideal}}[\Theta_a = \theta_a].$$

We derive each term as

$$\Pr_{\text{ideal}}[\Theta_a = \theta_a] = \Pr_{\text{ideal}}[K = k, \ell_i^\sharp = \mathsf{P}_1(N_i^\sharp), V_i^\sharp = \mathsf{P}_2(N_i^\sharp), i \in [\![c_N]\!]] = \frac{1}{2^n} \cdot \left(\frac{1}{(2^n)_{c_N}}\right)^2,$$

and

$$\Pr_{\text{ideal}}[\Theta_p = \theta_p | \Theta_a = \theta_a] = \Pr_{\text{ideal}}[E(\widehat{k}_i, \widehat{x}_i) = \widehat{y}_i, i \in [\![q_p]\!] | \Theta_a = \theta_a] = \prod_{j=1}^{\beta} \frac{1}{(2^n)_{\widehat{k}(j)}}.$$

Finally, we have

$$\begin{aligned}\Pr_{\text{ideal}}[\Theta_c = \theta_c | \Theta_p = \theta_p, \Theta_a = \theta_a] &= \Pr_{\text{ideal}}[\widetilde{\mathsf{P}}(t_i, x_i) = y_i, i \in [\![q_c]\!] | \Theta_p = \theta_p, \Theta_a = \theta_a] \\ &= \prod_{j=1}^{\alpha} \frac{1}{(2^n)_{t(j)}}.\end{aligned}$$

Transcript Probability in the Real World. With the same decomposition of transcript variables as before, we derive each term (here, a sampled value of L is written as $L = \ell$).

$$\Pr_{\text{real}}[\Theta_a = \theta_a] = \Pr_{\text{real}}[K = k, \ell_i^\sharp = E(k, N_i^\sharp), V_i^\sharp = E(k \oplus_{\shortparallel} 1, \ell_i^\sharp), i \in [\![c_N]\!]] = \frac{1}{2^n} \cdot \left(\frac{1}{(2^n)_{c_N}}\right)^2,$$

and

$$\Pr_{\text{real}}[\Theta_p = \theta_p | \Theta_a = \theta_a] = \Pr_{\text{real}}[E(\widehat{k}_i, \widehat{x}_i) = \widehat{y}_i, i \in [\![q_p]\!] | \Theta_a = \theta_a] = \prod_{j=1}^{\beta} \frac{1}{(2^n)_{\widehat{k}(j)}}.$$

Note that $\widehat{k}_i \notin \{k, k \oplus_{\shortparallel} 1\}$ holds from goodness of transcript. Finally, we have

$$\begin{aligned}\Pr_{\text{real}}[\Theta_c = \theta_c | \Theta_p = \theta_p, \Theta_a = \theta_a] &= \Pr_{\text{real}}[E(h_i^2, x_i) = y_i, i \in [\![q_c]\!] | \Theta_p = \theta_p, \Theta_a = \theta_a] \\ &= \prod_{j=1}^{\gamma} \frac{1}{(\Sigma(j))_{h^{2\sharp}(j)}}.\end{aligned}$$

We remark that $\Sigma(j)$ is fixed from $\mathcal{H}^1$ and $\mathcal{H}^2$, which is determined from the conditional clause. Also, being a good transcript, when $h_i^2 = h_j^2$ we have $x_i \neq x_j$, $y_i \neq y_j$.

Now it is easy to confirm the following proposition.

Proposition 3. *For any positive integer Q, $(Q)_{i+j} \leqslant (Q)_i \cdot (Q)_j$ for any i, j such that $i, j, i + j \in [\![Q]\!]$.*

We have

$$\prod_{j=1}^{\gamma} \frac{1}{(\Sigma(j))_{h^{2\sharp}(j)}} \geqslant \prod_{j=1}^{\alpha} \frac{1}{(2^n)_{t(j)}}.$$

This follows from Proposition 3 and the simple fact that a pair of different H^2 values must be created from a pair of different tweaks. Since the probability distribution of (Θ_p, Θ_a) is identical in both worlds, we have

$$\begin{aligned}\frac{\Pr_{\text{real}}[\Theta = \theta]}{\Pr_{\text{ideal}}[\Theta = \theta]} &= \frac{\Pr_{\text{real}}[\Theta_c = \theta_c | \Theta_p = \theta_p, \Theta_a = \theta_a]}{\Pr_{\text{ideal}}[\Theta_c = \theta_c | \Theta_p = \theta_p, \Theta_a = \theta_a]} \\ &= \prod_{j=1}^{\gamma} \frac{1}{(\Sigma(j))_{h^{2\sharp}(j)}} \cdot \prod_{j=1}^{\alpha} (2^n)_{t(j)} \geqslant 1. \qquad (15)\end{aligned}$$

From (14) and (15) and Lemma 3, we prove Lemma 2.

Proofs of Lemma 1, for ICE3. We have the following lemma for GICE2.
Lemma 4. *For (q_c, q_p)-adversary $\mathcal{A}$, we have*

$$\mathbf{Adv}^{\mathtt{tsprp}}_{\mathsf{GICE2}}(\mathcal{A}) \leqslant \frac{0.5q_c^2}{2^{k-8}} + \frac{q_c \cdot q_p}{2^k}.$$

We prove Lemma 4; it will also imply the proof of Lemma 1 for ICE3.

Since the essential proof ideas are shared with the proof of GICE1, we only briefly describe the proof of GICE2. We similarly define the transcript: there is no H^1 and $H^2 = 2^D L[1] \| L[2] \oplus B$, where $L = K \oplus (N \| 0^{k-nl})$ and $(L[1], L[2]) \overset{k-8}{\leftarrow} L$. Let $\overline{N}$ denote $N \| 0^{k-nl}$ and $(\overline{N}[1], \overline{N}[2]) \overset{k-8}{\leftarrow} \overline{N}$ and $(K[1], K[2]) \overset{k-8}{\leftarrow} K$ so that $L[1] = \overline{N}[1] \oplus K[1]$ and $L[2] = \overline{N}[2] \oplus K[2]$. In the ideal world, we use TURP as $C_i \leftarrow \widetilde{\mathsf{P}}(T_i, M_i)$ and random $K \overset{\$}{\leftarrow} \mathcal{K}$ (there is no additional URPs as we do not need KDF involving encryption). We observe that there are two bad events. The first is a key-input collision between two c-queries of different tweaks: $H_i^2 = H_j^2$ such that $T_i \neq T_j$ for $(i, j) \in [\![q_c]\!]^2$. The pairwise probability for this bad event is

$$\Pr_{\text{ideal}}[H_i^2 = H_j^2] = \Pr_{\text{ideal}}[2^{D_i} L_i[1] = 2^{D_j} L_j[1], L_i[2] \oplus L_j[2] = B_i \oplus B_j]. \tag{16}$$

We do a case analysis for $T_i \neq T_j$. When $D_i \neq D_j$, (16) is at most

$$\begin{aligned}\Pr_{\text{ideal}}[2^{D_i} L_i[1] = 2^{D_j} L_j[1]] &\leqslant \Pr_{\text{ideal}}[(2^{D_i} \oplus 2^{D_j}) K[1] = 2^{D_i} \overline{N_i}[1] \oplus 2^{D_j} \overline{N_j}[1]] \\ &\leqslant \frac{1}{2^{k-8}}\end{aligned}$$

from the property of doublings. When $D_i = D_j$ but $N_i \neq N_j$, it implies $\overline{N}_i[1] \neq \overline{N}_j[1]$, as we assumed $nl \leqslant k - 8$. Since $2^{D_i} L_i[1] \oplus 2^{D_j} L_j[1]$ equals $2^{D_i}(\overline{N}_i[1] \oplus \overline{N}_j[1]) \neq 0$ (i.e. a non-zero element of $\mathrm{GF}(2^{k-8})$), it implies (16) is 0.

Finally, when $D_i = D_j$, $N_i = N_j$, and $B_i \neq B_j$, because $L_i[2] \oplus L_j[2] = 0^8$, (16) is 0 too. Therefore, the first bad event has probability $\binom{q_c}{2}/2^{k-8} \leqslant 0.5q_c^2/2^{k-8}$.

The second bad event is a key-input collision between a c-query and a p-query. Since K is independent of the query responses, it is easy to see that

$$\Pr_{\text{ideal}}[2^{D_i} L_i[1] \| L_i[2] \oplus B_i = \widehat{K}_j] \leqslant \frac{1}{2^k}$$

for any $(i, j) \in [\![q_c]\!] \times [\![q_p]\!]$, hence the second bad event probability is $q_c \cdot q_p/2^k$.

Similarly to GICE1, we apply Lemma 3 and obtain $\epsilon_2 = 0.5q_c^2/2^{k-8} + q_c \cdot q_p/2^k$. We can also derive $\epsilon_1 = 0$ in a similar manner to the case of GICE1. Thus the advantage is at most $0.5q_c^2/2^{k-8} + q_c \cdot q_p/2^k$, which concludes the proof of Lemma 4.

A.3 Proofs of Remus-N (Theorem 4).

Idealized Remus-N. We reinterpret Remus-N as a mode of TBC $\widetilde{E}$, which is instantiated by ICE in the concrete specification. When the underlying TBC is TURP $\widetilde{\mathsf{P}}$, we write the mode as TRemus-N. It is similar to Romulus-N (with TURP) but has a different counter treatment and AD processing. See Figure 16.
Lemma 5. *For (q_e, q_d)-privacy-adversary $\mathcal{A}$ and (q_e, q_d)-authenticity adversary $\mathcal{B}$,*

$$\mathbf{Adv}^{\mathtt{priv}}_{\mathsf{TRemus\text{-}N}}(\mathcal{A}) = 0, \tag{17}$$

$$\mathbf{Adv}^{\mathtt{auth}}_{\mathsf{TRemus\text{-}N}}(\mathcal{B}) \leqslant \frac{3q_d}{2^n} + \frac{2q_d}{2^\tau}. \tag{18}$$

```
Algorithm TRemus-N.Enc$(N, A, M)$
  1. $S \leftarrow 0^n$
  2. $(A[1], \ldots, A[a]) \xleftarrow{n} A$
  3. $(C[1], \ldots, C[m]) \xleftarrow{n} C$
  4. if $|A[a]| < n$ then $w_A \leftarrow 13$ else $12$
  5. if $|C[m]| < n$ then $w_C \leftarrow 11$ else $10$
  6. $A[a] \leftarrow \mathtt{pad}_n(A[a])$
  7. for $i = 1$ to $a - 1$
  8.     $(S, \eta) \leftarrow \rho(S, A[i])$
  9.     $S \leftarrow \widetilde{\mathsf{P}}^{N,i,4}(S)$
 10. end for
 11. $(S, \eta) \leftarrow \rho(S, A[a])$
 12. $S \leftarrow \widetilde{\mathsf{P}}^{N,a,w_A}(S)$
 13. for $i = 1$ to $m - 1$
 14.     $(S, M[i]) \leftarrow \rho^{-1}(S, C[i])$
 15.     $S \leftarrow \widetilde{\mathsf{P}}^{N,a+i,2}(S)$
 16. end for
 17. $\widetilde{S} \leftarrow (0^{|C[m]|} \,\|\, \mathtt{rmt}_{n-|C[m]|}(G(S)))$
 18. $C'[m] \leftarrow \mathtt{pad}_n(C[m]) \oplus \widetilde{S}$
 19. $(S, M'[m]) \leftarrow \rho^{-1}(S, C'[m])$
 20. $M[m] \leftarrow \mathtt{lmt}_{|C[m]|}(M'[m])$
 21. $S \leftarrow \widetilde{\mathsf{P}}^{N,a+m,w_C}(S)$
 22. $(\eta, T^*) \leftarrow \rho(S, 0^n)$
 23. $M \leftarrow M[1] \,\|\, M[2] \,\|\, \ldots \,\|\, M[m]$
 24. if $T^* = T$ then return $M$ else $\perp$
```

```
Algorithm TRemus-N.Dec$(N, A, C, T)$
  1. $S \leftarrow 0^n$
  2. $(A[1], \ldots, A[a]) \xleftarrow{n} A$
  3. $(C[1], \ldots, C[m]) \xleftarrow{n} C$
  4. if $|A[a]| < n$ then $w_A \leftarrow 13$ else $12$
  5. if $|C[m]| < n$ then $w_C \leftarrow 11$ else $10$
  6. $A[a] \leftarrow \mathtt{pad}_n(A[a])$
  7. for $i = 1$ to $a - 1$
  8.     $(S, \eta) \leftarrow \rho(S, A[i])$
  9.     $S \leftarrow \widetilde{\mathsf{P}}^{N,i,4}(S)$
 10. end for
 11. $(S, \eta) \leftarrow \rho(S, A[a])$
 12. $S \leftarrow \widetilde{\mathsf{P}}^{N,a,w_A}(S)$
 13. for $i = 1$ to $m - 1$
 14.     $(S, M[i]) \leftarrow \rho^{-1}(S, C[i])$
 15.     $S \leftarrow \widetilde{\mathsf{P}}^{N,a+i,2}(S)$
 16. end for
 17. $\widetilde{S} \leftarrow (0^{|C[m]|} \,\|\, \mathtt{rmt}_{n-|C[m]|}(G(S)))$
 18. $C'[m] \leftarrow \mathtt{pad}_n(C[m]) \oplus \widetilde{S}$
 19. $(S, M'[m]) \leftarrow \rho^{-1}(S, C'[m])$
 20. $M[m] \leftarrow \mathtt{lmt}_{|C[m]|}(M'[m])$
 21. $S \leftarrow \widetilde{\mathsf{P}}^{N,a+m,w_C}(S)$
 22. $(\eta, T^*) \leftarrow \rho(S, 0^n)$
 23. $M \leftarrow M[1] \,\|\, M[2] \,\|\, \ldots \,\|\, M[m]$
 24. if $T^* = T$ then return $M$ else $\perp$
```

```
Algorithm $\rho(S, M)$
  1. $C \leftarrow M \oplus G(S)$
  2. $S' \leftarrow S \oplus M$
  3. return $(S', C)$
```

```
Algorithm $\rho^{-1}(S, C)$
  1. $M \leftarrow C \oplus G(S)$
  2. $S' \leftarrow S \oplus M$
  3. return $(S', M)$
```

Figure 16: A TBC-based abstraction of Remus-N. Here the TBC is a TURP $\widetilde{\mathsf{P}}$ having the same I/O as GICE1 and GICE2. Lines of [**if (statement) then** $X \leftarrow x$ **else** x'] are shorthand for [**if (statement) then** $X \leftarrow x$ **else** $X \leftarrow x'$]. The dummy variable η is always discarded. As an example, this abstracts Romulus-N1. In fact ρ has been redefined in the proof so that it directly accepts the final partial block and does padding internally (see Preliminaries at the beginning of this section).

Proof of Lemma 5. The proof of Lemma 5 is similar to those of (4) and (5).

Proof of Privacy Bound (17). With the same reasoning as Romulus-N, due to the uniqueness of the tweak values for all invoked $\widetilde{\mathsf{P}}$ calls, the privacy bound is zero.

Proof of Authenticity Bound (18). We first prove the case $q_d = 1$. Let (N', A', C', T') denote the single verification query (here we omit the subscript), and let p be the probability of successful forgery. Let Θ_e be the random variable representing the transcripts obtained by encryption queries and responses. By convention, we can assume $\mathcal{B}$ as deterministic (since the optimal adversary is always deterministic). Moreover, we only need to consider the case that $\mathcal{B}$ performs q_e encryption queries first and then one verification query.

Let a_i and m_i be the number of AD and plaintext blocks in i-th query (for Remus both are n-bit blocks, hence $a_i = |A_i|_n$ and $m_i = |M_i|_n$).

As in the proof of Romulus, we write $X_i[j]$ and $Y_i[j]$ to denote the j-th TURP input and output in the encryption, for $j \in [\![m_i]\!]$, where $Y_i[j]$ is to encrypt $M_i[j+1]$ when $j < m_i$ and $X_i[m_i]$ is given to $\widetilde{\mathsf{P}}$ with tweak $(N_i, w_{M_i}, a_i + m_i)$ to create $Y_i[m_i]$. Tag is $T_i = \mathtt{lmt}_\tau(\rho_2(Y_i[m_i], 0^n)) = \mathtt{lmt}_\tau(G(Y_i[m_i]))$. Similarly, let $V_i[j]$ and $W_i[j]$ be the j-th TURP input and output in the AD processing $j \in [\![a_i]\!]$. Let $S_i = W_i[a_i]$ be the state value after the AD process in i-th encryption query, which is also used to encrypt $M_i[1]$ if exists. Note that $X_i[1] = \rho_1(S_i, M_i[1])$.

Similarly, let a' and m' be AD and ciphertext block lengths of the single decryption query (N', A', C', T'), and define $X'[j]$, $Y'[j]$ and S' for the decryption query in the same fashion.

For such $\mathcal{B}$, we have

$$p = \Pr[T' = T^*] \leqslant \max_{\theta_e \in \mathcal{S}_{\Theta_e}} \Pr[T' = T^* | \Theta_e = \theta_e]$$

where $\mathcal{S}_{\Theta_e}$ denotes the set of all possible values for θ_e, and T^* denotes the true tag value for (N', A', C', T'). We also observe that

$$\Pr[\Theta_e] = \Pr[C^q, T^q, N^q, A^q, M^q]$$

and the probability $\Pr[C^q, T^q]$ is uniform and independent of the values of (N^q, A^q, M^q) from (1) as they are the TURP outputs taking distinct tweak.

We fix θ_e and write $\Pr_e$ to denote the conditional probability space given $\Theta_e = \theta_e$.

All $X_i[j]$ and $Y_i[j]$ are determined by Θ_e except $j = m_i$: they are cut when the last block is partial or $\tau < n$. For simplicity, we let encryption oracle to disclose $X_i[m_i]$ and $Y_i[m_i]$ and attach to Θ_e *after* the all encryption queries.

Let T^* be the true tag value for decryption query, i.e.

$$T^* = \mathtt{lmt}_\tau(G(Y'[m'])), \quad Y'[m'] = \widetilde{\mathsf{P}}^{(N', w_{C'}, a'+m')}(X'[m']), \tag{19}$$

where $w_{C'}$ is the indicator of padding for C'. As we assumed G is regular, if $Y'[m']$ is completely random from the previous variables (of transcript), the guessing probability of tag is $1/2^\tau$.

We do a case analysis.

Case 1: $N' \neq N_i$ for all i. Since the final tweak of decryption query is $(N', w_{C'}, a' + m')$ which has never been used before, from (19), $T^* = \mathtt{lmt}_\tau(G(Y'[m']))$ is unpredictable. $p_e = 1/2^\tau$.

Case 2: $N' = N_i$ for some $1 \leqslant i \leqslant q$. Wlog we let $N' = N_1$ and for simplicity omit the subscript i, i.e, (N, A, M, C, T) means $(N_1, A_1, M_1, C_1, T_1)$, and a and m mean a_1 and m_1. The same applies to $X_i[j]$ and $Y_i[j]$. When a variable X defined at encryption and X' defined at decryption are given $\widetilde{\mathsf{P}}$ of identical tweak (either thorough ρ or directly), we say X and X' are coupled. For example, when $a = a'$, $V[i]$ and $V'[i]$ for $i \in [\![a-1]\!]$ are coupled, and $V[a]$ and $V'[a]$ are coupled iff $w_A = w_{A'}$.

Case 2-1: $C' \neq C$, $(a+m, w_M) \neq (a'+m', w_{C'})$. Then the final tweak values, $(N, w_M, a+m)$ and $(N'(=N), w_{C'}, a'+m')$, are different, and the latter never appeared in the encryption queries. Thus $Y'[m']$ is random from (19), and we have $p_e = 1/2^\tau$.

Case 2-2: $C' \neq C$, $(a+m, w_M) = (a'+m', w_{C'})$, $a = a'$ (thus) $m = m'$. Then the analysis is the same as Case 2-2 of Romulus (note that S and S' are created by $\widetilde{\mathsf{P}}$ taking the identical tweak). We have $p_e \leqslant 2/2^\tau + 2/2^n$.

Case 2-3: $C' \neq C$, $(a+m, w_M) = (a'+m', w_{C'})$, $a \neq a'$ (thus) $m \neq m'$.

Case 2-3-1: $a' < a$. Let $b = a - a' > 0$, thus $m' = m + b$. We define δ for $0 \leqslant \delta \leqslant m$ as the block index of M (or C, i.e. message of encryption query) of the *last* difference between C and C', where we seek the difference from the last of ciphertext blocks. That is, when $\delta > 0$, $C[\delta] \neq C'[\delta + b]$ and $C[i] = C'[i+b]$ for any $\delta < i \leqslant m$. When $\delta = 0$ it means C is a postfix of C'. We remark that $C \neq C'$ comes from $b > 0$ (i.e. the block lengths are different). Note that the last X-values are $X[m]$ for encryption query and $X'[m+b]$ for decryption query and both go to $\widetilde{\mathsf{P}}^{(N, w_M, a+m)}$ to produce tags.

If $\delta > 1$, the analysis is the same as Case 2-2 of Romulus; more specifically, as C is not a postfix of C', we can apply the same analysis as Romulus on $C[\delta], \dots, C[m]$ and $C'[\delta+b], \dots, C'[m']$, where $C[\delta]$ and $C'[\delta+b]$ are coupled and so on, with case analysis on the difference between the coupled $X[\delta]$ and $X'[\delta+b]$.

If $\delta = 0$, it implies C is a postfix of C'. Then, the processing of $A[a]$ and that of $C[b]$ take different tweaks, (N, w_A, a) and (N, B, a) for B indicating "encryption" (e.g. $B = 2$ for Remus-N1), where the latter never appears in encryption. Thus $Y'[b]$ is random and independent of $W[a]$ which is coupled. The mapping $Y'[b] \to X'[m']$ and $W[a] \to X[m]$ are the same permutation, hence (in the same analysis as Case 2-2 of Romulus) we have $p_e = 2/2^\tau + 2/2^n$. The remaining case is $\delta = 1$. This is the same as the case $\delta = 0$: the different tweaks (N, w_A, a) and (N, B, a) make the coupled $X[1]$ and $X'[b+1]$ independent irrespective of the difference between $C[1]$ and $C'[b+1]$, and the mappings $X[1] \to X[m]$ and $X'[b+1] \to X'[m']$ are identical. Therefore, $p_e = 2/2^\tau + 2/2^n$ holds for the all cases of δ.

Case 2-3-2: $a' > a$. Let $b = a' - a > 0$, thus $m = m' + b$. The analysis is mostly symmetric to Case 2-3-1. We define δ in the same way: when $\delta > 1$ we have the same analysis as Case 2-3-1, and when $\delta = 1$ we claim $W'[a']$ is random and independent of $Y[b]$ which is coupled with $W'[a']$, while the latter process is the identical permutation. Thus $p_e = 2/2^\tau + 2/2^n$.

Case 3: $N' = N_i$ for some $i \in [\![q]\!]$, and $C' = C_i$ and (thus) $A' \neq A_i$. As in the analysis of Case 2, wlog we assume $i = 1$ and abbreviate A_i as A and so on. Note that $(m, w_M) = (m', w_{C'})$ holds.

Case 3-1: $a \neq a'$. It implies $a + m \neq a' + m'$ thus the final tweaks of encryption and decryption are different, thus $p_e = 1/2^n$ in this case.

Case 3-2: $a = a'$, thus $(a + m, w_M) = (a' + m', w_{C'})$. The analysis of this case is the same as Case 3 of Romulus. Specifically the analysis is obtained from Case 3-2 of Romulus, by setting all even AD blocks for both encryption and decryption to N. We note that we have fewer sub-cases than Romulus as an AD block never goes to tweak input in Remus. The difference between Romulus and TBC-based Remus in encryption part (only in the final tweak derivation and values of domain separation) does not change the proof, as Case 3-2 of Romulus only uses the fact that the encryption process is a permutation of S once we fix message and $\widetilde{\mathsf{P}}$. Thus, as in Case 3-2 of Romulus, $p_e = 2/2^\tau + 3/2^n$.

Therefore, for any fixed $\Theta_e = \theta_e$, $p_e \leqslant 3/2^n + 2/2^\tau$ holds for all cases. Thus the authenticity bound for $q_d = 1$ is $3/2^n + 2/2^\tau$. Following the generic conversion from $q_d = 1$ and $q_d \geqslant 1$ which multiplies q_d to the above, we conclude the proof of authenticity bound (18).

This concludes the proof of Lemma 5.

Deriving the Final Bounds. By combining Lemma 5 and Lemma 1, we have

$$\begin{aligned}
\mathbf{Adv}^{\mathtt{priv}}_{\mathsf{Remus\text{-}N1}}(\mathcal{A}) &\leqslant \mathbf{Adv}^{\mathtt{tsprp}}_{\mathsf{ICE1}}(\mathcal{A}') + \mathbf{Adv}^{\mathtt{priv}}_{\mathsf{TRemus\text{-}N}}(\mathcal{A}) \\
&\leqslant \left(\frac{9\sigma_{\mathrm{priv}}^2 + 4\sigma_{\mathrm{priv}} \cdot q_p}{2^n} + \frac{2q_p}{2^n}\right) + 0, \\
\mathbf{Adv}^{\mathtt{priv}}_{\mathsf{Remus\text{-}N2}}(\mathcal{A}) &\leqslant \mathbf{Adv}^{\mathtt{tsprp}}_{\mathsf{ICE2}}(\mathcal{A}') + \mathbf{Adv}^{\mathtt{priv}}_{\mathsf{TRemus\text{-}N}}(\mathcal{A}) \\
&\leqslant \left(\frac{9\sigma_{\mathrm{priv}}^2 + 4\sigma_{\mathrm{priv}} \cdot q_p}{2^{2n}} + \frac{2q_p}{2^n}\right) + 0, \\
\mathbf{Adv}^{\mathtt{priv}}_{\mathsf{Remus\text{-}N3}}(\mathcal{A}) &\leqslant \mathbf{Adv}^{\mathtt{tsprp}}_{\mathsf{ICE3}}(\mathcal{A}') + \mathbf{Adv}^{\mathtt{priv}}_{\mathsf{TRemus\text{-}N}}(\mathcal{A}) \\
&\leqslant \left(\frac{0.5\sigma_{\mathrm{priv}}^2}{2^{k-8}} + \frac{q_p \cdot \sigma_{\mathrm{priv}}}{2^k}\right) + 0
\end{aligned}$$

for privacy, where $\mathcal{A}'$ is $(\sigma_{\mathrm{priv}}, q_p)$-adversary. Similarly,

$$\begin{aligned}
\mathbf{Adv}^{\mathtt{auth}}_{\mathsf{Remus\text{-}N1}}(\mathcal{B}) &\leqslant \mathbf{Adv}^{\mathtt{tsprp}}_{\mathsf{ICE1}}(\mathcal{B}') + \mathbf{Adv}^{\mathtt{auth}}_{\mathsf{TRemus\text{-}N}}(\mathcal{B}) \\
&\leqslant \left(\frac{9\sigma_{\mathrm{auth}}^2 + 4\sigma_{\mathrm{auth}} \cdot q_p}{2^n} + \frac{2q_p}{2^n}\right) + \left(\frac{3q_d}{2^n} + \frac{2q_d}{2^\tau}\right), \\
\mathbf{Adv}^{\mathtt{auth}}_{\mathsf{Remus\text{-}N2}}(\mathcal{B}) &\leqslant \mathbf{Adv}^{\mathtt{tsprp}}_{\mathsf{ICE2}}(\mathcal{B}') + \mathbf{Adv}^{\mathtt{auth}}_{\mathsf{TRemus\text{-}N}}(\mathcal{B}) \\
&\leqslant \left(\frac{9\sigma_{\mathrm{auth}}^2 + 4\sigma_{\mathrm{auth}} \cdot q_p}{2^{2n}} + \frac{2q_p}{2^n}\right) + \left(\frac{3q_d}{2^n} + \frac{2q_d}{2^\tau}\right), \\
\mathbf{Adv}^{\mathtt{auth}}_{\mathsf{Remus\text{-}N3}}(\mathcal{B}) &\leqslant \mathbf{Adv}^{\mathtt{tsprp}}_{\mathsf{ICE3}}(\mathcal{B}') + \mathbf{Adv}^{\mathtt{auth}}_{\mathsf{TRemus\text{-}N}}(\mathcal{B}) \\
&\leqslant \left(\frac{0.5\sigma_{\mathrm{auth}}^2}{2^{k-8}} + \frac{q_p \cdot \sigma_{\mathrm{auth}}}{2^k}\right) + \left(\frac{3q_d}{2^n} + \frac{2q_d}{2^\tau}\right),
\end{aligned}$$

for authenticity, where $\mathcal{B}'$ is $(\sigma_{\mathrm{auth}}, q_p)$-adversary. This concludes the proof of Theorem 4.

A.4 Proofs of Romulus-M (Theorem 2 and Theorem 3)

We consider Romulus-M that uses a TURP $\widetilde{\mathsf{P}}$ as the underlying TBC.

Nonce-Respecting Privacy. It is easy to see that the privacy is perfect, since for an encryption query (N_i, A_i, M_i), all the TBC calls to compute the output (C_i, T_i) take distinct tweak values from the nonce-respecting assumption.

Nonce-Misusing Privacy. We start with analyzing a TURP $\widetilde{\mathsf{P}} : \overline{\mathcal{T}} \times \mathcal{M} \to \mathcal{M}$ with $\overline{\mathcal{T}} = \mathcal{T} \times \mathcal{B} \times \mathcal{D}$ used in Romulus-M. Let RF be a random function that has the same domain and range as $\widetilde{\mathsf{P}}$. Consider $\mathcal{A}$ that makes q queries and can repeat a tweak at most r times. We first claim that $\mathcal{A}$ has a small advantage in distinguishing between $\widetilde{\mathsf{P}}$ and RF. Let $\mathbf{Adv}^{\mathtt{prf}}_{\widetilde{\mathsf{P}}}(\mathcal{A}) = \Pr\left[\mathcal{A}^{\widetilde{\mathsf{P}}(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathsf{RF}(\cdot,\cdot)} \Rightarrow 1\right]$. In the real case, the oracle returns $C = \widetilde{\mathsf{P}}(T, M)$ for a TURP $\widetilde{\mathsf{P}}$. In the ideal case, the oracle returns $C = \mathsf{RF}(T, M)$ for a random function RF. $\mathcal{A}$ makes at most q queries and can use the same T at most r times. We have the following lemma.
Lemma 6. *Let $\mathcal{A}$ be an adversary as above. Then $\mathbf{Adv}^{\mathtt{prf}}_{\widetilde{\mathsf{P}}}(\mathcal{A}) \leqslant rq/2^n$.*

Proof. Consider the simulation of $\widetilde{\mathsf{P}}$ with the lazy-sampling, where we always return $C_i \xleftarrow{\$} \{0,1\}^n$ for the i-th query (T_i, M_i). For each $i \in [\![q]\!]$, there are at most r previous queries that share the same tweak T_i, and the simulation is successful if C_i does not collide with the corresponding r ciphertext blocks. That is, for each i, the simulation fails with probability at most $r/2^n$, and since $\mathcal{A}$ makes at most q queries, the result follows. □

In the encryption algorithm of Romulus-M in Figure 4, we refer to the mapping $(N, A, M) \mapsto T$ as the MAC part, and this corresponds to lines 1–24. We next show that the MAC part is a PRF. Let $\mathcal{A}$ be an adversary that makes q queries of the form (N, A, M), where the same N can be repeated at most r times. For (N, A, M), the MAC oracle, which we write MAC, returns T by following the definition of the MAC part. We define $\mathbf{Adv}^{\mathtt{prf}}_{\mathsf{MAC}}(\mathcal{A}) = \Pr\left[\mathcal{A}^{\mathsf{MAC}(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot,\cdot)} \Rightarrow 1\right]$, where \$ always returns $T \xleftarrow{\$} \{0,1\}^n$. We have the following lemma.
Lemma 7. *Let $\mathcal{A}$ be an adversary as above. Then $\mathbf{Adv}^{\mathtt{prf}}_{\mathsf{MAC}}(\mathcal{A}) \leqslant 4rq/2^n$.*

Proof. We without loss of generality assume that $\mathcal{A}$ makes q queries, and let $\mathcal{Q} = \{(N_1, A_1, M_1), \ldots, (N_q, A_q, M_q)\}$ be the set of queries. Consider the real case where we have the MAC oracle, and let $T_1, \ldots, T_q$ be the outputs of the oracle. We first replace all the $\widetilde{\mathsf{P}}^{(\cdot,\cdot,\cdot)}(\cdot)$ calls in line 23 of the MAC oracle with $\mathsf{RF}^{(\cdot,\cdot,\cdot)}(\cdot)$, i.e, all the last calls of $\widetilde{\mathsf{P}}$ to compute T_i is now replaced with a random function, and let MAC' be the resulting oracle. Since $\mathcal{A}$ can repeat the same nonce at most r times, we observe that $\widetilde{\mathsf{P}}$ in line 23 is called q times with the same tweak at most r times, so we use Lemma 6 to obtain

$$\mathbf{Adv}^{\mathtt{prf}}_{\mathsf{MAC}}(\mathcal{A}) \leqslant \frac{rq}{2^n} + \mathbf{Adv}^{\mathtt{prf}}_{\mathsf{MAC}'}(\mathcal{A}).$$

Consider two distinct inputs (N, A, M) and (N', A', M') of MAC', and suppose that we run lines 1–12. For (N, A, M), $X[1], \ldots, X[a]$ and $X[a+1], \ldots, X[a+m]$ are defined, and w is defined in line 10. The corresponding values $X'[1], \ldots, X'[a'+m']$ and w' are also defined for (N', A', M'). We say that, if $(N, w, a+m) = (N', w', a'+m')$, then (N, A, M) and (N', A', M') have the same *type*. We then run lines 13–24 to compute T and T', and let $S[a+m]$ be the input value of $\mathsf{RF}^{(N,w,a+m)}(\cdot)$ to compute T and $S'[a'+m']$ be the input value of $\mathsf{RF}^{(N',w',a'+m')}(\cdot)$ to compute T'. We observe that, if $S[a+m] \neq S'[a'+m']$ holds for any distinct $(N, A, M), (N', A', M') \in \mathcal{Q}$ with the same type, then the output distribution of MAC' is the same as that of \$.

We modify the definition of MAC' as follows. For the i-th query (N_i, A_i, M_i), we choose $T_i \xleftarrow{\$} \{0,1\}^n$, and return T_i to $\mathcal{A}$. All the q queries and responses are now fixed, and then we run lines 1–22 of the code of MAC'. We say that a bad event occurs if we have $S[a+m] = S'[a'+m']$ for some two distinct queries $(N, A, M), (N', A', M') \in \mathcal{Q}$ with the same type. We observe that unless the bad event occurs, this modification does not change the output distribution of MAC', since the random function RF can interpolate q input-output pairs with probability $1/2^{qn}$.

We use the following claim, which is proved after completing the proof of Lemma 7.

Claim. *For two distinct queries $(N, A, M), (N', A', M') \in \mathcal{Q}$ with the same type, we have* $\Pr[S[a+m] = S'[a'+m']] \leqslant 3/2^n$.

For each $i \in [\![q]\!]$, the i-th query (N_i, A_i, M_i) can cause the bad event with probability at most $3r/2^n$, since, among $(N_1, A_1, M_1), \ldots, (N_{i-1}, A_{i-1}, M_{i-1})$, there are at most r queries that have the same type as (N_i, A_i, M_i). Therefore, the probability of the bad event is at most $3rq/2^n$ as $\mathcal{A}$ makes at most q queries. This shows $\mathbf{Adv}^{\mathsf{prf}}_{\mathsf{MAC}'}(\mathcal{A}) \leqslant 3rq/2^n$, and we have the lemma. □

Proof of Claim. We follow the analysis in [NS20]. Let $p = \Pr[S[a+m] = S'[a'+m']]$. We use the notation defined in lines 1–22 of MAC', and let $S[2j-1]$ be the input of $\widetilde{\mathsf{P}}^{(X[2j],x,2j-1)}(\cdot)$ and $\widetilde{S}[2j-1]$ be the output. $S'[2j-1]$ and $\widetilde{S}'[2j-1]$ are defined analogously. We break the case as follows.

Case 1: $a = a'$ (and hence $m = m'$). Let δ be the largest index $\delta \in [\![a+m]\!]$ such that $X[\delta] \neq X'[\delta]$. We have $X[j] = X'[j]$ for all $\delta + 1 \leqslant j \leqslant a+m$.

If $\delta \geqslant 2$ is even, we observe that $S[a+m] = S'[a+m]$ holds iff $\widetilde{S}[\delta-1] = \widetilde{S}'[\delta-1]$, since the function that maps $\widetilde{S}[\delta-1]$ to $S[a+m]$ is a permutation over $\{0,1\}^n$ from the soundness of G. Therefore,

$$p = \Pr[\widetilde{S}[\delta-1] = \widetilde{S}'[\delta-1]]$$

holds. Since $\widetilde{S}'[\delta-1]$ is uniformly random over $\{0,1\}^n$, we obtain $p = 1/2^n$ in this case.

If $\delta = 1$, we must have $S[1] \neq S'[1]$, and this implies $p = 0$.

If $\delta \geqslant 3$ is odd, we proceed as follows.

$$\begin{aligned} p \leqslant\ & \Pr[S[a+m] = S'[a+m] \mid S[\delta-2] = S'[\delta-2] \wedge X[\delta-1] = X'[\delta-1]] \\ &+ \Pr[S[a+m] = S'[a+m] \mid S[\delta-2] = S'[\delta-2] \wedge X[\delta-1] \neq X'[\delta-1]] \\ &+ \Pr[S[a+m] = S'[a+m] \mid S[\delta-2] \neq S'[\delta-2]] \end{aligned}$$

The first term is 0, since we have $S[\delta] \neq S'[\delta]$ and this ensures $S[a+m] \neq S'[a+m]$. The second term is at most $1/2^n$, since $\widetilde{S}'[\delta-2]$ is uniformly random over $\{0,1\}^n$. The third term is at most $1/(2^n-1)$, since $\widetilde{S}'[\delta-2]$ is uniformly random over $\{0,1\}^n$ or over $\{0,1\}^n \backslash \{\widetilde{S}[\delta-2]\}$. Overall, we have

$$p \leqslant 0 + \frac{1}{2^n} + \frac{1}{2^n-1} \leqslant \frac{3}{2^n}$$

when $\delta \geqslant 3$, and for Case 1.

Case 2: $a \neq a'$ (and hence $m \neq m'$). Let δ be the largest index $\delta \in [\![a+m]\!]$ such that $X[\delta] \neq X'[\delta]$.

Let $a_{\max} = \max\{a, a'\}$. Note that we have $a_{\max} \geqslant 2$ since we have $a \geqslant a'+1 \geqslant 2$ or $a' \geqslant a+1 \geqslant 2$, and a and a' cannot be 0 from the definition of the alternating parsing. There are two cases to consider.

- If $\delta \in \{a_{\max}+1, \ldots, a+m\}$, then we can follow the same analysis as in Case 1, and we obtain $p \leqslant 3/2^n$.
- If $\delta \in [\![a_{\max}]\!]$, then we have the following unique tweaks depending on a and a':
  - $(A[a], 40, a-1)$ (if a is even and $a \geqslant a'+1$)
  - $(A[a-1], 40, a-2)$ (if a is odd and $a \geqslant a'+1$)
  - $(A'[a'], 40, a'-1)$ (if a' is even and $a' \geqslant a+1$)
  - $(A'[a'-1], 40, a'-2)$ (if a' is odd and $a' \geqslant a+1$)

  In either case, the output of $\widetilde{\mathsf{P}}$ with the above tweak is uniformly random over $\{0,1\}^n$, and we have $p \leqslant 1/2^n$.

Combining Cases 1 and 2, we have $p \leqslant 3/2^n$ for any case. □

Next, we focus on the analysis of the encryption part: $(N, M) \mapsto (C, T)$, where $T \xleftarrow{\$} \{0,1\}^n$ is chosen uniformly at random, and C is computed with $\widetilde{\mathsf{P}}$ as in [PS16]. More precisely, let ENC be an oracle that takes (N, M) as input, internally chooses $T \xleftarrow{\$} \{0,1\}^n$, and returns (C, T) by executing lines 25–36 of the encryption algorithm of Romulus-M in Figure 4. Let $\mathcal{A}$ be an adversary that makes q queries (N_i, M_i), where the same N_i can be repeated at most r times, and the total block length of M_i is at most σ blocks, i.e., if $(M_i[1], \ldots, M_i[m_i]) \xleftarrow{n} M_i$, then $m_1 + \cdots + m_q \leqslant \sigma$. For (N_i, M_i), the encryption oracle, which we write ENC, returns (C_i, T_i) by following the steps stated above. We define $\mathbf{Adv}^{\mathtt{priv\$}}_{\mathsf{ENC}}(\mathcal{A}) = \Pr\left[\mathcal{A}^{\mathsf{ENC}(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot)} \Rightarrow 1\right]$, where \$ always returns $(C_i, T_i) \xleftarrow{\$} \{0,1\}^{|M_i|+n}$. We have the following lemma.

Lemma 8. *Let $\mathcal{A}$ be an adversary as above. Then $\mathbf{Adv}^{\mathtt{priv\$}}_{\mathsf{ENC}}(\mathcal{A}) \leqslant 2r\sigma/2^n$.*

Proof. We first replace all the calls to $\widetilde{\mathsf{P}}$ in line 31 with a random function RF, and let ENC' be the resulting encryption scheme. Observe that $\widetilde{\mathsf{P}}$ is called at most σ times in total, and the same tweak is used at most r times. We can therefore use Lemma 6 to obtain

$$\mathbf{Adv}^{\mathtt{priv\$}}_{\mathsf{ENC}}(\mathcal{A}) \leqslant \frac{r\sigma}{2^n} + \mathbf{Adv}^{\mathtt{priv\$}}_{\mathsf{ENC}'}(\mathcal{A}).$$

For (N_i, M_i), let $S_i[j]$ be the input value of $\mathsf{RF}^{(N_i,36,j)}(\cdot)$ for $0 \leqslant j \leqslant m_i - 1$. Note that $S_i[0] = T_i$. For each $i \in [\![q]\!]$, we say that a bad event occurs if there exist $i' \in [\![i-1]\!]$ and $0 \leqslant j \leqslant \min\{m_i - 1, m_{i'} - 1\}$ such that $N_{i'} = N_i$ and $S_{i'}[j] = S_i[j]$. That is, the bad event occurs if there exists a previous query $(N_{i'}, M_{i'})$ that has the same nonce as (N_i, M_i) and if one of the inputs of $\mathsf{RF}^{(N_i,36,j)}(\cdot)$ with the same index j collides. It is easy to see that unless the bad event occurs, the output distribution of ENC' is the same as that of \$. We see that the i-th query (N_i, M_i) can cause the bad event with probability at most $rm_i/2^n$, and hence we obtain

$$\mathbf{Adv}^{\mathtt{priv\$}}_{\mathsf{ENC}'}(\mathcal{A}) \leqslant \sum_{i \in [\![q]\!]} \frac{rm_i}{2^n} \leqslant \frac{r\sigma}{2^n},$$

and this concludes the proof of Lemma 8. □

Finally, we consider the privacy of $\mathsf{Romulus\text{-}M}$. Recall that we are analyzing $\mathcal{A}$ that makes at most q_e encryption queries, can repeat a nonce at most r times, and the total effective blocks is at most σ_{priv} blocks.

We see that $\mathsf{Romulus\text{-}M}$ can be described as follows. For (N, A, M), we let $T \leftarrow \mathsf{MAC}(N, A, M)$ and $C \leftarrow \mathsf{ENC}(N, M)$, where we use T as the internal randomness of ENC, and return (C, T).

Let ρ be a random function that has the same domain and range as MAC, and let $\mathsf{Romulus\text{-}M}'$ be $\mathsf{Romulus\text{-}M}$ where we use ρ instead of MAC. From Lemma 7, we have

$$\mathbf{Adv}^{\texttt{nm-priv}}_{\mathsf{Romulus\text{-}M}}(\mathcal{A}) \leqslant \frac{3rq_e}{2^n} + \mathbf{Adv}^{\texttt{nm-priv}}_{\mathsf{Romulus\text{-}M}'}(\mathcal{A}).$$

Given (N, A, M), T is uniformly random in $\mathsf{Romulus\text{-}M}'$, and hence we can use Lemma 8 to obtain

$$\mathbf{Adv}^{\texttt{nm-priv}}_{\mathsf{Romulus\text{-}M}'}(\mathcal{A}) \leqslant \frac{r\sigma_{\text{priv}}}{2^n},$$

and this gives $\mathbf{Adv}^{\texttt{nm-priv}}_{\mathsf{Romulus\text{-}M}}(\mathcal{A}) \leqslant 3rq_e/2^n + 3r\sigma_{\text{priv}}/2^n \leqslant 4r\sigma_{\text{priv}}/2^n$ in Theorem 3.

Nonce-Respecting Authenticity. Without loss of generality, we assume that $\mathcal{B}$ makes q_e encryption queries, and does not repeat a query.

We give $\mathcal{B}$ direct access to an oracle $\widetilde{\mathsf{P}}^{(\cdot,36,\cdot)}(\cdot)$, i.e., $\mathcal{B}$ can now compute $Z = \widetilde{\mathsf{P}}^{(N,36,i)}(X)$ for any (N, i, X) of its choice. This only increases the advantage of $\mathcal{B}$, and observe that for an encryption query (N_i, A_i, M_i), $\mathcal{B}$ can compute C_i from T_i and the oracle $\widetilde{\mathsf{P}}^{(\cdot,36,\cdot)}(\cdot)$, and that for a decryption query (N', A', C', T'), $\mathcal{B}$ can compute M' from T', C', and the oracle $\widetilde{\mathsf{P}}^{(\cdot,36,\cdot)}(\cdot)$. We now modify the game as follows:

- For an encryption query (N_i, A_i, M_i), we only return T_i.
- Instead of making a decryption query (N', A', C', T'), we ask $\mathcal{B}$ to make a verification query of the form (N', A', M', T').

With this modification, we can focus on the analysis of the MAC part. Note that we call (N', A', M', T') a verification query (instead of a decryption query).

For an encryption query (N_i, A_i, M_i), we run lines 1–24 of the encryption algorithm and return T_i, and $\mathcal{B}$ can compute C_i by using the oracle $\widetilde{\mathsf{P}}^{(\cdot,36,\cdot)}(\cdot)$. For a decryption query (N', A', C', T'), $\mathcal{B}$ can compute the corresponding M' by using $\widetilde{\mathsf{P}}^{(\cdot,36,\cdot)}(\cdot)$, and for a verification query (N', A', M', T'), we run lines 1–24 of the *encryption* algorithm and return 1, indicating $\mathcal{B}$ wins the game, iff $T^* = T'$, where T^* is defined in line 24. Let p be the probability of successful forgery of $\mathcal{B}$ in this game.

We first consider $\mathcal{B}$ that makes single verification query (N', A', M', T'), i.e., $q_d = 1$. We can assume that $\mathcal{B}$ is deterministic, and we only need to consider the case that $\mathcal{B}$ makes q_e encryption queries first and then makes one verification query.

Let (N', A', M', T') be the verification query of a' AD blocks and m' message blocks, i.e., $(A'[1], \ldots, A'[a']) \xleftarrow{n,t} A'$ and $(M'[1], \ldots, M'[m']) \xleftarrow{n,t} M'$ (or $(M'[1], \ldots, M'[m']) \xleftarrow{t,n} M'$). We do a case analysis.

Case 1: $(N', w', a'+m') \neq (N_i, w_i, a_i+m_i)$ for all i. Since the final tweak of the verification query is $(N', w', a' + m')$, which has never been used before, T^* is unpredictable, and we have $p = 1/2^n$.

Case 2: $(N', w', a' + m') = (N_i, w_i, a_i + m_i)$ for some $i \in [\![q]\!]$. The analysis of this case is almost identical to the proof of Claim in Lemma 7. From the nonce-respecting assumption, $(N_i, w_i, a_i + m_i)$ is unique, and we omit the subscript and write $(N, w, a + m)$ for $(N_i, w_i, a_i + m_i)$. Let $S'[a' + m']$ be the input value of $\widetilde{\mathsf{P}}^{(N',w',a'+m')}(\cdot)$ to compute T^* and $S[a + m]$ be the input value to compute T.

Now in this case, $\widetilde{\mathsf{P}}$ to compute $T^* = G(\widetilde{\mathsf{P}}^{(N',w',a'+m')}(S'[a' + m']))$ takes the same tweak as the tweak to compute $T = G(\widetilde{\mathsf{P}}^{(N,w,a+m)}(S[a + m]))$, and $\mathcal{A}$ wins iff $T^* = T'$.

We follow the notation defined in lines 1–24 of the encryption algorithm, and let $S'[2j - 1]$ be the input of $\widetilde{\mathsf{P}}^{(X'[2j],x,2j-1)}(\cdot)$ and $\widetilde{S}'[2j - 1]$ be the output. $S[2j - 1]$ and $\widetilde{S}[2j - 1]$ are defined analogously. We further break the case as follows.

Case 2-1: $a' = a$ (and $m' = m$). Let δ be the largest index $\delta \in [\![a' + m']\!]$ such that $X'[\delta] \neq X[\delta]$. We have $X'[j] = X[j]$ for all $\delta + 1 \leqslant j \leqslant a' + m'$.

If $\delta \geqslant 2$ is even, we have $S'[a' + m'] = S[a' + m']$ iff $\widetilde{S}'[\delta - 1] = \widetilde{S}[\delta - 1]$, since the function that maps $\widetilde{S}'[\delta - 1]$ to $S'[a' + m']$ is a permutation over $\{0, 1\}^n$, and that

$$p \leqslant \frac{2}{2^n} + \Pr[\widetilde{S}'[\delta - 1] = \widetilde{S}[\delta - 1]]$$

by following the same analysis as Romulus-N. Since $\widetilde{S}'[\delta - 1]$ is uniformly random over $\{0, 1\}^n$, we obtain $p \leqslant 2/2^n + 1/2^n = 3/3^n$ in this case.

If $\delta = 1$, we must have $S'[1] \neq S[1]$, and this implies $p \leqslant 2/2^n$.

If $\delta \geqslant 3$ is odd, we proceed as follows.

$$\begin{aligned}
&\Pr[S'[a' + m'] = S[a' + m']] \\
&\leqslant \Pr[S'[a' + m'] = S[a' + m'] \mid S'[\delta - 2] = S[\delta - 2] \wedge X'[\delta - 1] = X[\delta - 1]] \\
&+ \Pr[S'[a' + m'] = S[a' + m'] \mid S'[\delta - 2] = S[\delta - 2] \wedge X'[\delta - 1] \neq X[\delta - 1]] \\
&+ \Pr[S'[a' + m'] = S[a' + m'] \mid S'[\delta - 2] \neq S[\delta - 2]]
\end{aligned}$$

The first term is 0, since $S'[\delta] \neq S[\delta]$ ensures $S'[a' + m'] \neq S[a' + m']$. The second term is at most $1/2^n$, since $\widetilde{S}'[\delta - 2]$ is uniformly random over $\{0, 1\}^n$. The third term is at most $1/(2^n - 1)$, since $\widetilde{S}'[\delta - 2]$ is uniformly random over $\{0, 1\}^n$ or over $\{0, 1\}^n \backslash \{\widetilde{S}[\delta - 2]\}$. Overall, we have

$$p \leqslant \frac{2}{2^n} + 0 + \frac{1}{2^n} + \frac{1}{2^n - 1} \leqslant \frac{5}{2^n}$$

for Case 2-1.

Case 2-2: $a' \neq a$ (and hence $m' \neq m$). Let δ be the largest index $\delta \in [\![a' + m']\!]$ such that $X'[\delta] \neq X[\delta]$. Let $a_{\max} = \max\{a', a\}$.

- If $\delta \in \{a_{\max} + 1, \ldots, a' + m'\}$, then we follow the same analysis as in Case 2-1, and we obtain $p \leqslant \frac{5}{2^n}$.
- If $\delta \in [\![a_{\max}]\!]$, then we have the following unique tweaks depending on a' and a:
  - $(A'[a'], 40, a' - 1)$ (if a' is even and $a' \geqslant a + 1$)
  - $(A'[a' - 1], 40, a' - 2)$ (if a' is odd and $a' \geqslant a + 1$)
  - $(M'[a - a'], 44, a - 1)$ (if a is even and $a' \leqslant a - 1$)

– $(M'[a - a' - 1], 44, a - 2)$ (if a is odd and $a' \leqslant a - 1$)

In either case, the output of $\widetilde{\mathsf{P}}$ with the above tweak is uniformly random over $\{0,1\}^n$, and we have $p \leqslant 2/2^n + 1/2^n = 3/2^n$ for Case 2-2.

Therefore, we have $p \leqslant 5/2^n$ for any case.

When the adversary can make q_d decryption queries, we use the generic conversion to obtain $p \leqslant 5q_d/2^n$ in Theorem 2.

Nonce-Misusing Authenticity. We give direct access to an oracle $\widetilde{\mathsf{P}}^{(\cdot,36,\cdot)}(\cdot)$ to the adversary as in the nonce-respecting case. Let p be the probability of successful forgery of $\mathcal{B}$ in the modified game, and we consider the game based on the MAC part. We note that the MAC part is very similar to the NaT construction in [CLS17], and we follow its analysis. We write the MAC part as $T \leftarrow \mathsf{MAC}(N, A, M)$, and the proof is based on Patarin's Coefficient-H technique [Pat08]. Our adversary $\mathcal{B}$ makes q_e encryption queries (N_i, A_i, M_i) and q_d verification queries (N'_j, A'_j, M'_j, T'_j), and we consider a distinguishing game between the real world $(\mathsf{MAC}, \mathsf{Ver})$ and the ideal world $(\mathsf{Rand}, \mathsf{Rej})$.

In the real world, for an encryption query (N_i, A_i, M_i), the MAC oracle computes $T_i \leftarrow \mathsf{MAC}(N_i, A_i, M_i)$ following the specification and returns T_i to $\mathcal{B}$, and for a decryption query (N'_j, A'_j, M'_j, T'_j), the Ver oracle computes $T^*_j = \mathsf{MAC}(N'_j, A'_j, M'_j)$, and returns $\bot$ if $T'_j \neq T^*_j$. Otherwise Ver returns $\top$ to $\mathcal{B}$. In the ideal world, Rand oracle returns $T_i \xleftarrow{\$} \{0,1\}^n$ to $\mathcal{B}$, and for a decryption query, Rej returns $\bot$. Let

$$p^* = \Pr\left[\mathcal{B}^{\mathsf{MAC},\mathsf{Ver}} \Rightarrow 1\right] - \Pr\left[\mathcal{B}^{\mathsf{Rand},\mathsf{Rej}} \Rightarrow 1\right].$$

We have $p \leqslant p^*$, and we focus on the analysis of p^*.

In the real world, we execute lines 1–24 of the encryption algorithm of $\mathsf{Romulus\text{-}M}$, and let K_h be all the "keys" used to execute lines 1–22, i.e., the randomness of $\widetilde{\mathsf{P}}$ used to compute the tags except for the last calls. In the ideal world, K_h is generated randomly following the real world by the simulation of $\widetilde{\mathsf{P}}$, i.e., K_h is generated exactly as in the real world. We disclose K_h to $\mathcal{B}$ after making all the encryption and verification queries but before it outputs the decision bit. This only increases the advantage of $\mathcal{B}$.

Let $\tau = (\tau_e, \tau_v, K_h)$ be all the queries of $\mathcal{B}$ and responses from the oracles, and additional information that $\mathcal{B}$ gains, which we call the transcript, where

$$\begin{cases} \tau_e = ((N_1, A_1, M_1, T_1), \ldots, (N_{q_e}, A_{q_e}, M_{q_e}, T_{q_e})), \\ \tau_v = ((N'_1, A'_1, M'_1, T'_1, b'_1), \ldots, (N'_{q_d}, A'_{q_d}, M'_{q_d}, T'_{q_d}, b'_{q_d})). \end{cases}$$

Here, $b'_i \in \{\top, \bot\}$, and it holds that $b'_i = \bot$ for all i in the ideal world. A transcript is attainable if the probability to obtain the transcript in the ideal world is non-zero, and let Θ be the set of all attainable transcripts. We let Θ_{ideal} and Θ_{real} denote the probability distributions of the transcript in the ideal world and in the real world, respectively. Based on these notation, we restate the Coefficient-H technique [Pat08] as follows.

Lemma 9. *For a distinguisher $\mathcal{B}$, let $\Theta = \mathsf{GoodT} \cup \mathsf{BadT}$ be a partition of the set of all attainable transcripts. If there exist ϵ_1 and ϵ_2 such that for any $\tau \in \mathsf{GoodT}$,*

$$\frac{\Pr[\Theta_{\text{real}} = \tau]}{\Pr[\Theta_{\text{ideal}} = \tau]} \geqslant 1 - \epsilon_1,$$

and $\Pr[\Theta_{\text{ideal}} \in \mathsf{BadT}] \leqslant \epsilon_2$, then the distinguishing advantage of $\mathcal{B}$ is bounded by $\epsilon_1 + \epsilon_2$.

For an encryption query (N_i, A_i, M_i), let $S_i[a_i + m_i]$ be the input state of $\widetilde{\mathsf{P}}^{(N_i,w_i,a_i+m_i)}$ in line 23. Similarly, for a verification query (N'_j, A'_j, M'_j, T'_j), let $S'_j[a'_j + m'_j]$ be the input state of $\widetilde{\mathsf{P}}^{(N'_j,w'_j,a'_j+m'_j)}$ in line 23. We say that two encryption queries (N_i, A_i, M_i) and (N_j, A_j, M_j) have the same type if $(N_i, w_i, a_i + m_i) = (N_j, w_j, a_j + m_j)$. Similarly, we say that an encryption query (N_i, A_i, M_i) and a verification query (N'_j, A'_j, M'_j, T'_j) have the same type if $(N_i, w_i, a_i + m_i) = (N'_j, w'_j, a'_j + m'_j)$.

We are now ready to define a bad transcript. We say that $\tau = (\tau_e, \tau_v, K_h)$ is bad if

- there exist two encryption queries (N_i, A_i, M_i) and (N_j, A_j, M_j) with the same type such that $S_i[a_i + m_i] = S_j[a_j + m_j]$ or $T_i = T_j$, or
- there exist an encryption query (N_i, A_i, M_i) and a verification query (N'_j, A'_j, M'_j, T'_j) with the same type such that $S_i[a_i + m_i] = S'_j[a'_j + m'_j]$ and $T_i = T'_j$.

Otherwise we say that a transcript is good. Note that the second event cannot happen in the real world, since the verification oracle returns $\top$ for the query (N'_j, A'_j, M'_j, T'_j), and this is impossible for an attainable transcript. Let BadT be the set of bad transcripts, and GoodT be the set of good transcripts.

We first show that, in the ideal world, the probability of obtaining the bad transcript is small.

Lemma 10. $\Pr[\Theta_{\text{ideal}} \in \mathsf{BadT}] \leqslant \dfrac{4rq_e}{2^n} + \dfrac{3rq_d}{2^n}$.

We first consider the first condition. Fix (N_i, A_i, M_i), where we have q_e choices. We then fix (N_j, A_j, M_j) that has the same type as (N_i, A_i, M_i). We have at most $r - 1$ choices. We have $\Pr[T_i = T_j] = 1/2^n$ from the randomness of T_i and T_j. We also have $\Pr[S_i[a_i + m_i] = S'_j[a'_j + m'_j]] \leqslant 3/2^n$ from the analysis of Cases 2-1 and 2-2 of the nonce-respecting case, or the Claim in the proof of Lemma 7 in the nonce-misusing privacy proof. Therefore, the first condition can occur with probability at most

$$\frac{(r-1)q_e}{2^n} + \frac{3(r-1)q_e}{2^n} = \frac{4(r-1)q_e}{2^n}. \tag{20}$$

For the second condition, we fix a verification query (N'_j, A'_j, M'_j, T'_j), where we have q_d choices, and there are at most r encryption queries that have the same type as the verification query. Let (N_i, A_i, M_i) be one of them.

- If (N'_j, A'_j, M'_j, T'_j) is queried after (N_i, A_i, M_i), then we must have $T_i \neq T'_j$, or $(A_i, M_i) \neq (A'_j, M'_j)$. If $T_i \neq T'_j$, then the condition cannot be satisfied. If $(A_i, M_i) \neq (A'_j, M'_j)$, then we have $\Pr[S_i[a_i + m_i] = S'_j[a'_j + m'_j]] \leqslant 3/2^n$ as above.
- If (N'_j, A'_j, M'_j, T'_j) is queried before (N_i, A_i, M_i), then we have $\Pr[T_i = T'_j] = 1/2^n$.

Overall, the second condition can occur with probability at most

$$\frac{3rq_d}{2^n}, \tag{21}$$

and we obtain the lemma from (20) and (21).

We next show that the ratio of the interpolation probabilities is close to one.

Lemma 11. $\dfrac{\Pr[\Theta_{\text{real}} = \tau]}{\Pr[\Theta_{\text{ideal}} = \tau]} \geqslant 1 - \dfrac{2q_d}{2^n}$.

Fix a transcript $\tau \in \mathsf{GoodT}$. Let $((N_1, w_1, a_1 + m_1), \ldots, (N_{q_e}, w_{q_e}, a_{q_e} + m_{q_e}))$ be a list of the q_e nonces used in encryption queries and associated tweak values to determine the type of the queries. Let μ_i be the multiplicity of $(N_i, w_i, a_i + m_i)$, i.e., the number of times that $(N_i, w_i, a_i + m_i)$ appears in the list. Note that we have $\mu_i \leqslant r$ for $1 \leqslant i \leqslant q_e$. We also let R be the number of representative elements of the list, i.e., R is the number of elements when we see $\{(N_1, w_1, a_1 + m_1), \ldots, (N_{q_e}, w_{q_e}, a_{q_e} + m_{q_e})\}$ as a set. Without loss of generality, we may assume that $\mu_1, \ldots, \mu_R$ represent the multiplicity of representative elements.

In the ideal world, the Rand oracle returns a random string and the Rej oracle returns $\perp$, and hence we have

$$\Pr[\Theta_{\text{ideal}} = \tau] = \Pr[\mathcal{K}_h = K_h] \cdot \frac{1}{(2^n)^{q_e}},$$

where $\Pr[\mathcal{K}_h = K_h]$ denotes the probability over $\widetilde{\mathsf{P}}$ that we obtain K_h as the randomness to execute lines 1–22. Note that in the ideal world, K_h is generated by following the same procedure as in the real world.

In the real world, the following equalities and inequalities have to be satisfied.

$$\mathsf{MAC}(N_i, A_i, M_i) = T_i \text{ for all } i \in [\![q_e]\!], \tag{22}$$

$$\mathsf{MAC}(N'_j, A'_j, M'_j) \neq T'_j \text{ for all } j \in [\![q_d]\!], \tag{23}$$

where we say that $\widetilde{\mathsf{P}}$ is compatible with (τ_e, K_h) if all the equalities in (22) are satisfied, and $\widetilde{\mathsf{P}}$ is compatible with (τ_v, K_h) if all the inequalities in (23) are satisfied. We say that $\widetilde{\mathsf{P}}$ is compatible with $\tau = (\tau_e, \tau_v, K_h)$ if it is compatible with (τ_e, K_h) and (τ_v, K_h). Let $\mathsf{Comp}(\tau)$ be the set of all $\widetilde{\mathsf{P}}$'s that are compatible with τ. Then we have

$$\Pr[\Theta_{\text{real}} = \tau] = \Pr[\mathcal{K}_h = K_h] \cdot \Pr[\widetilde{\mathsf{P}} \in \mathsf{Comp}(\tau)],$$

where $\Pr[\mathcal{K}_h = K_h]$ denotes the probability over $\widetilde{\mathsf{P}}$ that we obtain K_h as the randomness to execute lines 1–22, and the last probability is taken over $\widetilde{\mathsf{P}}$ that induces K_h. Now by following exactly the same argument as in [CLS17], we obtain

$$\Pr[\widetilde{\mathsf{P}} \in \mathsf{Comp}(\tau)] \geqslant \frac{1}{\prod_{i \in [\![R]\!]} (2^n)_{\mu_i}} \left(1 - \frac{q_d}{2^n - r}\right). \tag{24}$$

To see this, we define

$$\lambda_{\text{eq}} = \{(Tw_1, S_1[a_1 + m_1], T_1), \ldots, (Tw_{q_e}, S_{q_e}[a_{q_e} + m_{q_e}], T_{q_e})\},$$
$$\lambda_{\text{ineq}} = \{(Tw'_1, S'_1[a'_1 + m'_1], T'_1), \ldots, (Tw'_{q_d}, S'_{q_d}[a'_{q_d} + m'_{q_d}], T'_{q_d})\}$$

where $Tw_i = (N_i, w_i, a_i + m_i)$ for $i \in [\![q_e]\!]$ and $Tw'_j = (N'_j, w'_j, a'_j + m'_j)$ for $j \in [\![q_d]\!]$. Here, λ_{eq} is a permutation equalities list, and λ_{ineq} is a permutation inequalities list. That is, we require $\widetilde{\mathsf{P}}^{Tw_i}(S_i[a_i + m_i]) = T_i$ holds for all $i \in [\![q_e]\!]$ and $\widetilde{\mathsf{P}}^{Tw'_j}(S'_j[a'_j + m'_j]) \neq T'_j$ holds for all $j \in [\![q_d]\!]$. We also have $\lambda_{\text{eq}} \cap \lambda_{\text{ineq}} = \varnothing$, and [CLS17, Lemma 3] gives (24).

We can now compute the ratio as

$$\frac{\Pr[\Theta_{\text{real}} = \tau]}{\Pr[\Theta_{\text{ideal}} = \tau]} \geqslant \left(1 - \frac{q_d}{2^n - r}\right) \prod_{i \in [\![R]\!]} \frac{(2^n)^{\mu_i}}{(2^n)_{\mu_i}} \geqslant 1 - \frac{2q_d}{2^n} \tag{25}$$

from $(2^n)^{\mu_i}/(2^n)_{\mu_i} \geqslant 1$ and $1 \leqslant r \leqslant 2^{n-1}$.

Overall, from Lemma 9, Lemma 11, and (25), we have

$$p^* \leqslant \frac{4rq_e}{2^n} + \frac{3rq_d}{2^n} + \frac{2q_d}{2^n} \leqslant \frac{4rq_e}{2^n} + \frac{5rq_d}{2^n}$$

in Theorem 3.

A.5 Proofs of Remus-M (Theorem 5 and Theorem 6)

We start with defining the abstraction of TRemus-M based on a TURP $\widetilde{\mathsf{P}}$ that has the same I/O as GICE1. TRemus-M is defined in Figure 17, and we first show the following lemma.

Lemma 12. *For (q_e, σ_{priv})-privacy-adversary $\mathcal{A}$ and (q_e, q_d)-authenticity adversary $\mathcal{B}$, we have*

$$\begin{aligned}
\mathbf{Adv}^{\mathtt{priv}}_{\mathsf{TRemus\text{-}M}}(\mathcal{A}) &= 0,\\
\mathbf{Adv}^{\mathtt{nm\text{-}priv}}_{\mathsf{TRemus\text{-}M}}(\mathcal{A}) &\leqslant \frac{4r\sigma_{priv}}{2^n},\\
\mathbf{Adv}^{\mathtt{auth}}_{\mathsf{TRemus\text{-}M}}(\mathcal{B}) &\leqslant \frac{5q_d}{2^n},\\
\mathbf{Adv}^{\mathtt{nm\text{-}auth}}_{\mathsf{TRemus\text{-}M}}(\mathcal{B}) &\leqslant \frac{4rq_e}{2^n} + \frac{5rq_d}{2^n},
\end{aligned}$$

where the adversaries can repeat a nonce in encryption queries at most r times in the nonce-misuse cases.

The proofs are similar to those of Romulus-M.

Nonce-Respecting Privacy. We see that for an encryption query (N_i, A_i, M_i), all the TBC calls to compute the output (C_i, T_i) take distinct tweak values from the nonce-respecting assumption. Therefore, the privacy is perfect.

Nonce-Misusing Privacy. Similarly to the analysis of Romulus-M, we define the MAC part and encryption part. In the encryption algorithm of TRemus-M in Figure 17, the MAC part corresponds to lines 1–20, and is a mapping $(N, A, M) \mapsto T$.

We show that the MAC part is a PRF. Let $\mathcal{A}$ be an adversary that makes q queries and can repeat a nonce at most r times. Let MAC be an oracle that implements the MAC part, and let $\mathbf{Adv}^{\mathtt{prf}}_{\mathsf{MAC}}(\mathcal{A}) = \Pr\left[\mathcal{A}^{\mathsf{MAC}(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot,\cdot)} \Rightarrow 1\right]$, where \$ always returns $T \xleftarrow{\$} \{0,1\}^n$. We have the following lemma.

Lemma 13. *Let $\mathcal{A}$ be an adversary as above. Then $\mathbf{Adv}^{\mathtt{prf}}_{\mathsf{MAC}}(\mathcal{A}) \leqslant 4rq/2^n$.*

Proof. Wlog, assume that $\mathcal{A}$ makes q queries, and let $\mathcal{Q} = \{(N_1, A_1, M_1), \ldots, (N_q, A_q, M_q)\}$ be the set of queries. Suppose that $\mathcal{A}$ has MAC as the oracle, and we first replace all the $\widetilde{\mathsf{P}}^{(\cdot,\cdot,\cdot)}(\cdot)$ calls in line 19 with $\mathsf{RF}^{(\cdot,\cdot,\cdot)}(\cdot)$, which is a random function that has the same domain and range as $\widetilde{\mathsf{P}}$. Let MAC′ be the resulting oracle. We use Lemma 6 to obtain

$$\mathbf{Adv}^{\mathtt{prf}}_{\mathsf{MAC}}(\mathcal{A}) \leqslant \frac{rq}{2^n} + \mathbf{Adv}^{\mathtt{prf}}_{\mathsf{MAC}'}(\mathcal{A}).$$

For two distinct inputs (N, A, M) and (N', A', M') of MAC′, we say that, if $(N, a + m, w_M) = (N', a' + m', w'_M)$, then (N, A, M) and (N', A', M') have the same *type*. Let $S[a+m]$ be the input of $\mathsf{RF}^{(N,a+m,w_M)}(\cdot)$ and $S'[a'+m']$ be the input of $\mathsf{RF}^{(N',a'+m',w'_M)}(\cdot)$. If $S[a+m] \neq S'[a'+m']$ holds for any distinct $(N, A, M), (N', A', M') \in \mathcal{Q}$ with the same type, then the output distribution of MAC′ is the same as that of \$.

We modify MAC′ as follows. For the i-th query (N_i, A_i, M_i), we return $T_i \xleftarrow{\$} \{0,1\}^n$. Then q queries and responses are fixed. A bad event is defined as the event that $S[a+m] = S'[a'+m']$ for some two distinct queries $(N, A, M), (N', A', M') \in \mathcal{Q}$ with the same type.

```
Algorithm TRemus-M.Enc_K(N, A, M)
  1. $S \leftarrow 0^n$
  2. $(A[1], \ldots, A[a]) \xleftarrow{n} A$
  3. $(M[1], \ldots, M[m]) \xleftarrow{n} M$
  4. if $|A[a]| < n$ then $w_A \leftarrow 45$ else $44$
  5. if $|M[m]| < n$ then $w_M \leftarrow 47$ else $46$
  6. $A[a] \leftarrow \mathtt{pad}_n(A[a])$
  7. for $i = 1$ to $a - 1$
  8.     $(S, \eta) \leftarrow \rho(S, A[i])$
  9.     $S \leftarrow \widetilde{\mathsf{P}}^{N,i,36}(S)$
 10. end for
 11. $(S, \eta) \leftarrow \rho(S, A[a])$
 12. $S \leftarrow \widetilde{\mathsf{P}}^{N,a,w_A}(S)$
 13. for $i = 1$ to $m - 1$
 14.     $(S, \eta) \leftarrow \rho(S, M[i])$
 15.     $S \leftarrow \widetilde{\mathsf{P}}^{N,a+i,38}(S)$
 16. end for
 17. $M'[m] \leftarrow \mathtt{pad}_n(M[m])$
 18. $(S, \eta) \leftarrow \rho(S, M'[m])$
 19. $S \leftarrow \widetilde{\mathsf{P}}^{N,a+m,w_M}(S)$
 20. $(\eta, T) \leftarrow \rho(S, 0^n)$
 21. if $M = \epsilon$ then return $(\epsilon, T)$
 22. $S \leftarrow T$
 23. for $i = 1$ to $m - 1$
 24.     $S \leftarrow \widetilde{\mathsf{P}}^{N,i-1,34}(S)$
 25.     $(S, C[i]) \leftarrow \rho(S, M[i])$
 26. end for
 27. $S \leftarrow \widetilde{\mathsf{P}}^{N,m-1,34}(S)$
 28. $(\eta, C'[m]) \leftarrow \rho(S, M'[m])$
 29. $C[m] \leftarrow \mathtt{lsb}_{|M[m]|}(C'[m])$
 30. $C \leftarrow C[1] \,\|\, C[2] \,\|\, \ldots \,\|\, C[m]$
 31. return $(C, T)$
```

```
Algorithm TRemus-M.Dec_K(N, A, C, T)
  1. if $C = \epsilon$ then $M \leftarrow \epsilon$
  2. else
  3.     $S \leftarrow T$
  4.     $(C[1], \ldots, C[m]) \xleftarrow{n} C$
  5.     $z \leftarrow |C[m]|$
  6.     $C[m] \leftarrow \mathtt{pad}_n(C[m])$
  7.     for $i = 1$ to $m$
  8.         $S \leftarrow \widetilde{\mathsf{P}}^{N,i-1,34}(S)$
  9.         $(S, M[i]) \leftarrow \rho^{-1}(S, C[i])$
 10.     end for
 11.     $M[m] \leftarrow \mathtt{lsb}_z(M[m])$
 12.     $M \leftarrow M[1] \,\|\, \ldots \,\|\, M[m]$
 13. $S \leftarrow 0^n$
 14. $(A[1], \ldots, A[a]) \xleftarrow{n} A$
 15. if $|A[a]| < n$ then $w_A \leftarrow 45$ else $44$
 16. if $|M[m]| < n$ then $w_M \leftarrow 47$ else $46$
 17. $A[a] \leftarrow \mathtt{pad}_n(A[a])$
 18. for $i = 1$ to $a - 1$
 19.     $(S, \eta) \leftarrow \rho(S, A[i])$
 20.     $S \leftarrow \widetilde{\mathsf{P}}^{N,i,36}(S)$
 21. end for
 22. $(S, \eta) \leftarrow \rho(S, A[a])$
 23. $S \leftarrow \widetilde{\mathsf{P}}^{N,a,w_A}(S)$
 24. for $i = 1$ to $m - 1$
 25.     $(S, \eta) \leftarrow \rho(S, M[i])$
 26.     $S \leftarrow \widetilde{\mathsf{P}}^{N,a+i,38}(S)$
 27. end for
 28. $M'[m] \leftarrow \mathtt{pad}_n(M[m])$
 29. $(S, \eta) \leftarrow \rho(S, M'[m])$
 30. $S \leftarrow \widetilde{\mathsf{P}}^{N,a+m,w_M}(S)$
 31. $(\eta, T^*) \leftarrow \rho(S, 0^n)$
 32. if $T^* = T$ then return $M$ else $\perp$
```

```
Algorithm $\rho(S, M)$
  1. $C \leftarrow M \oplus G(S)$
  2. $S' \leftarrow S \oplus M$
  3. return $(S', C)$
```

```
Algorithm $\rho^{-1}(S, C)$
  1. $M \leftarrow C \oplus G(S)$
  2. $S' \leftarrow S \oplus M$
  3. return $(S', M)$
```

Figure 17: Encryption and decryption of TRemus-M, a TBC-based abstraction of Remus-M. It uses a TBC $\widetilde{\mathsf{P}}$ that has the same I/O as GICE1. Remus-M1 is used as a working example.

Unless the bad event occurs, this modification does not change the output distribution of MAC'.

We have the following claim. The proof is given after completing the proof of Lemma 13.

Claim. *For two distinct queries $(N, A, M), (N', A', M') \in \mathcal{Q}$ with the same type, we have $\Pr[S[a+m] = S'[a'+m']] \leqslant 3/2^n$.*

Now with the same reasoning as in the proof of Lemma 7, the probability of the bad event is at most $3rq/2^n$, and we have the lemma. □

Proof of Claim. Again, we rely on the approach in [NS20]. Let $p = \Pr[S[a+m] = S'[a'+m']]$. Let $S[j]$ be the input of $\widetilde{\mathsf{P}}^{(\cdot,j,\cdot)}(\cdot)$ and $\widetilde{S}[j]$ be the output. We also define $S'[j]$ and $\widetilde{S}'[j]$. For $(A[1], \ldots, A[a]) \overset{n}{\leftarrow} A$ and $(M[1], \ldots, M[m]) \overset{n}{\leftarrow} M$, let $X = (X[1], \ldots, X[a+m]) = (A[1], \ldots, \mathtt{pad}_n(A[a]), M[1], \ldots, \mathtt{pad}_n(M[m]))$, and let δ be the largest index $\delta \in [\![a+m]\!]$ such that $X[\delta] \neq X'[\delta]$. We break the case as follows.

Case 1: $a = a'$ (and hence $m = m'$) and $w_A = w'_A$. If $\delta = 1$, we must have $S[1] \neq S'[1]$, and this implies $p = 0$.

If $\delta \geqslant 2$, we proceed as follows.

$$\begin{aligned} p \leqslant\ & \Pr[S[a+m] = S'[a+m] \mid S[\delta-1] = S'[\delta-1] \wedge T[\delta-1] = T'[\delta-1]] \\ &+ \Pr[S[a+m] = S'[a+m] \mid S[\delta-1] = S'[\delta-1] \wedge T[\delta-1] \neq T'[\delta-1]] \\ &+ \Pr[S[a+m] = S'[a+m] \mid S[\delta-1] \neq S'[\delta-1]] \end{aligned}$$

Here, $T[\delta-1]$ denotes the tweak value of $\widetilde{\mathsf{P}}^{(\cdot,\delta-1,\cdot)}$ for (N, A, M), and $T'[\delta-1]$ is that for (N', A', M'). The first term is 0, since $S[\delta] \neq S'[\delta]$ holds and this implies $S[a+m] \neq S'[a+m]$. The second term is at most $1/2^n$, since $\widetilde{S}'[\delta-1]$ is uniformly random over $\{0,1\}^n$. The third term is at most $1/(2^n-1)$, since $\widetilde{S}'[\delta-1]$ is uniformly random over $\{0,1\}^n$ or over $\{0,1\}^n \backslash \{\widetilde{S}[\delta-1]\}$.

Overall, we have

$$p \leqslant 0 + \frac{1}{2^n} + \frac{1}{2^n - 1} \leqslant \frac{3}{2^n}$$

when $\delta \geqslant 2$, and for Case 1.

Case 2: $a = a'$ (and hence $m = m'$) and $w_A \neq w'_A$. If $\delta \in \{a+1, \ldots, a+m\}$, then the analysis of Case 1 applies, and we have $p \leqslant 3/2^n$. If $\delta \in [\![a]\!]$, then $T'[a]$ is unique and hence $\widetilde{S}'[a]$ is uniformly random over $\{0,1\}^n$. We have $p \leqslant 1/2^n$.

Overall, we have $p \leqslant 3/2^n$ for Case 2.

Case 3: $a \neq a'$ (and hence $m \neq m'$). Let $a_{\max} = \max\{a, a'\}$. There are two cases to consider.

- If $\delta \in \{a_{\max}+1, \ldots, a+m\}$, then we can follow the same analysis as in Case 1, and we obtain $p \leqslant 3/2^n$.
- If $\delta \in [\![a_{\max}]\!]$, then we have the following unique tweaks depending on a and a':
  - (N, a, w_A) (if $a \geqslant a'+1$)
  - (N', a', w'_A) (if $a' \geqslant a+1$)

  In either case, the output of $\widetilde{\mathsf{P}}$ with the above tweak is uniformly random over $\{0,1\}^n$, and we have $p \leqslant 1/2^n$.

By taking the maximum of Cases 1–3, we have $p \leqslant 3/2^n$ for any case. □

Next, we analyze the encryption part: $(N, M) \mapsto (C, T)$, where $T \xleftarrow{\$} \{0,1\}^n$ is chosen uniformly at random, and C is computed with $\widetilde{\mathsf{P}}$. We see that the encryption part of TRemus-M is the same as that of Romulus-M with minor differences in the tweak values. We define ENC, $\mathbf{Adv}^{\mathtt{priv\$}}_{\mathsf{ENC}}(\mathcal{A})$, and the adversary $\mathcal{A}$ as in Romulus-M, and we have the following lemma.

Lemma 14. *Let $\mathcal{A}$ be an adversary as in Lemma 8. Then $\mathbf{Adv}^{\mathtt{priv\$}}_{\mathsf{ENC}}(\mathcal{A}) \leqslant 2r\sigma/2^n$.*

We now consider the privacy of TRemus-M. Consider $\mathcal{A}$ that makes at most q_e queries, can repeat a nonce at most r times, and the total number of effective blocks is at most σ_{priv} blocks.

Then with the same analysis as Romulus-M, we obtain $\mathbf{Adv}^{\mathtt{nm\text{-}priv}}_{\mathsf{TRemus\text{-}M}}(\mathcal{A}) \leqslant 3rq_e/2^n + 3r\sigma_{\mathrm{priv}}/2^n \leqslant 4r\sigma_{\mathrm{priv}}/2^n$ in Lemma 12.

Nonce-Respecting Authenticity. As in the analysis of Romulus-M, we give $\mathcal{B}$ the direct access to an oracle $\widetilde{\mathsf{P}}^{(\cdot,\cdot,34)}(\cdot)$, and modify the game as follows:

- For an encryption query (N_i, A_i, M_i), we only return T_i.
- Instead of making a decryption query (N', A', C', T'), we ask $\mathcal{B}$ to make a verification query of the form (N', A', M', T').

We focus on the analysis of the MAC part. Let p be the probability of successful forgery of $\mathcal{B}$ in this game.

We first focus on the case $q_d = 1$, and consider $\mathcal{B}$ that makes single verification query (N', A', M', T').

We assume that $\mathcal{B}$ is deterministic, and that $\mathcal{B}$ makes q_e encryption queries first and then makes one verification query.

We do a case analysis. Let (N', A', M', T') be the verification query of a' AD blocks and m' message blocks.

Case 1: $(N', a'+m', w'_M) \neq (N_i, a_i+m_i, w_{M\,i})$ for all i. The final tweak of the verification query $(N', a'+m', w'_M)$ was not used before, T^* is uniformly random over $\{0,1\}^n$, and we have $p = 1/2^n$.

Case 2: $(N', a'+m', w'_M) = (N_i, a_i+m_i, w_{M\,i})$ for some $i \in [\![q_e]\!]$. From the nonce-respecting assumption, $(N_i, a_i+m_i, w_{M\,i})$ is unique, and we write $(N, a+m, w_M)$ for it. Let $S'[a'+m']$ be the input value of $\widetilde{\mathsf{P}}^{(N',a'+m',w'_M)}(\cdot)$, and $S[a+m]$ be the input value of $\widetilde{\mathsf{P}}^{(N,a+m,w_M)}(\cdot)$. Also, we let $S'[j]$ be the input of $\widetilde{\mathsf{P}}^{(\cdot,j,\cdot)}(\cdot)$ for (N', A', M'), and $\widetilde{S}'[j]$ be the output. $S[j]$ and $\widetilde{S}[j]$ are defined analogously. Let δ be the largest index $\delta \in [\![a'+m']\!]$ such that $X'[\delta] \neq X[\delta]$. We have $X'[j] = X[j]$ for all $\delta+1 \leqslant j \leqslant a'+m'$. Note that $X' = (X'[1], \ldots, X'[a'+m']) = (A'[1], \ldots, \mathtt{pad}_n(A'[a']), M'[1], \ldots, \mathtt{pad}_n(M'[m']))$, and similarly for X. We further break the case as follows.

Case 2-1: $a' = a$ (and $m' = m$) and $w'_A = w_A$. By following the analysis of Romulus-N, we have

$$p \leqslant \frac{2}{2^n} + \Pr[S'[\delta] = S[\delta]].$$

If $\delta = 1$, we must have $S'[1] \neq S[1]$, and this implies $p \leqslant 2/2^n$.

If $\delta \geqslant 2$, then we have

$$\begin{aligned}
&\Pr[S'[a'+m'] = S[a'+m']] \\
&\leqslant \Pr[S'[a'+m'] = S[a'+m'] \mid S'[\delta-1] = S[\delta-1] \wedge T'[\delta-1] = T[\delta-1]] \\
&+ \Pr[S'[a'+m'] = S[a'+m'] \mid S'[\delta-1] = S[\delta-1] \wedge T'[\delta-1] \neq T[\delta-1]] \\
&+ \Pr[S'[a'+m'] = S[a'+m'] \mid S'[\delta-1] \neq S[\delta-1]],
\end{aligned}$$

where $T'[\delta-1]$ is the tweak value of $\widetilde{\mathsf{P}}^{(\cdot,\delta-1,\cdot)}$ for (N', A', M'), and $T[\delta-1]$ is that for (N, A, M). The first term is 0 from $S'[\delta] \neq S[\delta]$ which ensures $S'[a'+m'] \neq S[a'+m']$. The second term is at most $1/2^n$, since $\widetilde{S}'[\delta-1]$ is uniformly random over $\{0,1\}^n$. The third term is at most $1/(2^n-1)$, since $\widetilde{S}'[\delta-1]$ is uniformly random over $\{0,1\}^n$ or over $\{0,1\}^n \backslash \{\widetilde{S}[\delta-1]\}$. Overall, we have

$$p \leqslant \frac{2}{2^n} + 0 + \frac{1}{2^n} + \frac{1}{2^n-1} \leqslant \frac{5}{2^n}$$

for Case 2-1.

Case 2-2: $a' = a$ (and hence $m' = m$) and $w'_A \neq w_A$. If $\delta \in \{a'+1, \ldots, a'+m'\}$, then with the same analysis as in Case 2-1, and we have $p \leqslant 5/2^n$. If $\delta \in [\![a']\!]$, then $T'[a']$ is unique and hence $\widetilde{S}'[a']$ is uniformly random over $\{0,1\}^n$. We have $p \leqslant 3/2^n$.

Overall, we have $p \leqslant 3/2^n$ for Case 2-2.

Case 2-3: $a' \neq a$ (and hence $m' \neq m$). Let $a_{\max} = \max\{a', a\}$.

- If $\delta \in \{a_{\max}+1, \ldots, a'+m'\}$, then we follow the same analysis as in Case 2-1, and we obtain $p \leqslant \frac{5}{2^n}$.
- If $\delta \in [\![a_{\max}]\!]$, then we have the following unique tweaks:
  - $(N', 38, a)$ (if $a \geqslant a'+1$)
  - (N', w'_A, a') (if $a' \geqslant a+1$)

  In either case, the output of $\widetilde{\mathsf{P}}$ with the above tweak is uniformly random over $\{0,1\}^n$, and we have $p \leqslant 2/2^n + 1/2^n = 3/2^n$ for Case 2-3.

Therefore, we have $p \leqslant 5/2^n$ for any case.

When the adversary can make q_d decryption queries, we use the standard conversion to obtain $p \leqslant 5q_d/2^n$ in Lemma 12.

Nonce-Misusing Authenticity. The proof is almost identical to that of Romulus-M. We give the direct access to an oracle $\widetilde{\mathsf{P}}^{(\cdot,34,\cdot)}(\cdot)$ to the adversary, and let p be the probability of successful forgery of $\mathcal{B}$ in the modified game of the MAC part, which is defined as in the nonce-respecting case. $\mathcal{B}$ makes q_e encryption queries and q_d verification queries.

For the i-th encryption query (N_i, A_i, M_i), we compute $T_i \leftarrow \mathsf{MAC}(N_i, A_i, M_i)$, and return T_i to $\mathcal{B}$. Then $\mathcal{B}$ outputs (N'_j, A'_j, M'_j, T'_j) for $j \in [\![q_d]\!]$, and $\mathcal{B}$ wins if $T'_j = T^*_j$, where $T^*_j = \mathsf{MAC}(N'_j, A'_j, M'_j)$.

We modify this game into a distinguishing game as in the analysis of Romulus-M. In the real world, for an encryption query, the MAC oracle computes $T_i \leftarrow \mathsf{MAC}(N_i, A_i, M_i)$ following the specification and returns T_i to $\mathcal{B}$, and for a verification query, the Ver oracle computes $T^*_j = \mathsf{MAC}(N'_j, A'_j, M'_j)$, and returns $\perp$ if $T'_j \neq T^*_j$. Otherwise Ver returns $\top$ to

$\mathcal{B}$. In the ideal world, Rand oracle returns $T_i \xleftarrow{\$} \{0,1\}^n$ to $\mathcal{B}$, and for a decryption query, Rej returns $\perp$. Let

$$p^* = \Pr\left[\mathcal{B}^{\mathsf{MAC},\mathsf{Ver}} \Rightarrow 1\right] - \Pr\left[\mathcal{B}^{\mathsf{Rand},\mathsf{Rej}} \Rightarrow 1\right].$$

We have $p \leqslant p^*$, where p is the success probability in the original game, and we focus on the analysis of p^*.

For two encryption queries (N_i, A_i, M_i) and (N_j, A_j, M_j), we say that they share the same type if the tweak values $(N_i, a_i + m_i, w_{M\,i})$ and $(N_j, a_j + m_j, w_{M\,j})$ used in the final $\widetilde{\mathsf{P}}$ calls are the same. Similarly, for an encryption query (N_i, A_i, M_i) and a verification query (N'_j, A'_j, M'_j, T'_j), they share the same type if $(N_i, a_i + m_i, w_{M\,i}) = (N'_j, a'_j + m'_j, w_{M}{}'_j)$.

We see that the reset of the analysis of p^* follows that of $\mathsf{Romulus\text{-}M}$, where the difference is the details of the evaluation of $\Pr[S_i[a_i+m_i] = S_j[a_j+m_j]]$, where $S_i[a_i+m_i]$ denotes the input value of $\widetilde{\mathsf{P}}^{(N_i, a_i+m_i, w_{M\,i})}(\cdot)$ for an encryption query (N_i, A_i, M_i), and $\Pr[S_i[a_i+m_i] = S'_j[a'_j + m'_j]]$, where $S'_j[a'_j + m'_j]$ denotes the input value of $\widetilde{\mathsf{P}}^{(N'_j, a'_j+m'_j, w_{M}{}'_j)}(\cdot)$ for a verification query (N'_j, A'_j, M'_j, T'_j), which can be both bounded from above by $3/2^n$ from the analysis of the nonce-respecting case.

Overall, we have

$$p^* \leqslant \frac{4rq_e}{2^n} + \frac{3rq_d}{2^n} + \frac{2q_d}{2^n} \leqslant \frac{4rq_e}{2^n} + \frac{5rq_d}{2^n}$$

in Lemma 12.

Deriving the Final Bounds. In the nonce-respecting case, by combining Lemma 12 and Lemma 1, we have

$$\begin{aligned}
\mathbf{Adv}^{\mathtt{priv}}_{\mathsf{Remus\text{-}M1}}(\mathcal{A}) &\leqslant \mathbf{Adv}^{\mathtt{tsprp}}_{\mathsf{ICE1}}(\mathcal{A}') + \mathbf{Adv}^{\mathtt{priv}}_{\mathsf{TRemus\text{-}M}}(\mathcal{A}) \\
&\leqslant \left(\frac{9\sigma^2_{\mathrm{priv}} + 4q_p\sigma_{\mathrm{priv}}}{2^n} + \frac{2q_p}{2^n}\right) + 0, \\
\mathbf{Adv}^{\mathtt{priv}}_{\mathsf{Remus\text{-}M2}}(\mathcal{A}) &\leqslant \mathbf{Adv}^{\mathtt{tsprp}}_{\mathsf{ICE2}}(\mathcal{A}') + \mathbf{Adv}^{\mathtt{priv}}_{\mathsf{TRemus\text{-}M}}(\mathcal{A}) \\
&\leqslant \left(\frac{9\sigma^2_{\mathrm{priv}} + 4q_p\sigma_{\mathrm{priv}}}{2^{2n}} + \frac{2q_p}{2^n}\right) + 0
\end{aligned}$$

for privacy, where $\mathcal{A}'$ is $(\sigma_{\mathrm{priv}}, q_p)$-adversary. Similarly,

$$\begin{aligned}
\mathbf{Adv}^{\mathtt{auth}}_{\mathsf{Remus\text{-}M1}}(\mathcal{B}) &\leqslant \mathbf{Adv}^{\mathtt{tsprp}}_{\mathsf{ICE1}}(\mathcal{B}') + \mathbf{Adv}^{\mathtt{auth}}_{\mathsf{TRemus\text{-}M}}(\mathcal{B}) \\
&\leqslant \left(\frac{9\sigma^2_{\mathrm{auth}} + 4q_p\sigma_{\mathrm{auth}}}{2^n} + \frac{2q_p}{2^n}\right) + \frac{5q_d}{2^n}, \\
\mathbf{Adv}^{\mathtt{auth}}_{\mathsf{Remus\text{-}M2}}(\mathcal{B}) &\leqslant \mathbf{Adv}^{\mathtt{tsprp}}_{\mathsf{ICE2}}(\mathcal{B}') + \mathbf{Adv}^{\mathtt{auth}}_{\mathsf{TRemus\text{-}M}}(\mathcal{B}) \\
&\leqslant \left(\frac{9\sigma^2_{\mathrm{auth}} + 4q_p\sigma_{\mathrm{auth}}}{2^{2n}} + \frac{2q_p}{2^n}\right) + \frac{5q_d}{2^n},
\end{aligned}$$

for authenticity, where $\mathcal{B}'$ is $(\sigma_{\text{auth}}, q_p)$-adversary. In the nonce-misusing case, we have

$$\begin{aligned}
\mathbf{Adv}^{\texttt{nm-priv}}_{\textsf{Remus-M1}}(\mathcal{A}) &\leqslant \mathbf{Adv}^{\texttt{tsprp}}_{\textsf{ICE1}}(\mathcal{A}') + \mathbf{Adv}^{\texttt{nm-priv}}_{\textsf{TRemus-M}}(\mathcal{A}) \\
&\leqslant \left(\frac{9\sigma_{\text{priv}}^2 + 4q_p\sigma_{\text{priv}}}{2^n} + \frac{2q_p}{2^n}\right) + \frac{4r\sigma_{\text{priv}}}{2^n}, \\
\mathbf{Adv}^{\texttt{nm-priv}}_{\textsf{Remus-M2}}(\mathcal{A}) &\leqslant \mathbf{Adv}^{\texttt{tsprp}}_{\textsf{ICE2}}(\mathcal{A}') + \mathbf{Adv}^{\texttt{nm-priv}}_{\textsf{TRemus-M}}(\mathcal{A}) \\
&\leqslant \left(\frac{9\sigma_{\text{priv}}^2 + 4q_p\sigma_{\text{priv}}}{2^{2n}} + \frac{2q_p}{2^n}\right) + \frac{4r\sigma_{\text{priv}}}{2^n}
\end{aligned}$$

for privacy, where $\mathcal{A}'$ is $(\sigma_{\text{priv}}, q_p)$-adversary. Similarly,

$$\begin{aligned}
\mathbf{Adv}^{\texttt{nm-auth}}_{\textsf{Remus-M1}}(\mathcal{B}) &\leqslant \mathbf{Adv}^{\texttt{tsprp}}_{\textsf{ICE1}}(\mathcal{B}') + \mathbf{Adv}^{\texttt{nm-auth}}_{\textsf{TRemus-M}}(\mathcal{B}) \\
&\leqslant \left(\frac{9\sigma_{\text{auth}}^2 + 4q_p\sigma_{\text{auth}}}{2^n} + \frac{2q_p}{2^n}\right) + \left(\frac{4rq_e}{2^n} + \frac{5rq_d}{2^n}\right), \\
\mathbf{Adv}^{\texttt{nm-auth}}_{\textsf{Remus-M2}}(\mathcal{B}) &\leqslant \mathbf{Adv}^{\texttt{tsprp}}_{\textsf{ICE2}}(\mathcal{B}') + \mathbf{Adv}^{\texttt{nm-auth}}_{\textsf{TRemus-M}}(\mathcal{B}) \\
&\leqslant \left(\frac{9\sigma_{\text{auth}}^2 + 4q_p\sigma_{\text{auth}}}{2^{2n}} + \frac{2q_p}{2^n}\right) + \left(\frac{4rq_e}{2^n} + \frac{5rq_d}{2^n}\right),
\end{aligned}$$

for authenticity, where $\mathcal{B}'$ is $(\sigma_{\text{auth}}, q_p)$-adversary. This concludes the proof of Theorems 5 and 6.

Table 9: Members of Romulus.

Family	Name	$\widetilde{E}$	k	nl	n	t	d	τ
Romulus-N	Romulus-N1	Skinny-128-384	128	128	128	128	56	128
	Romulus-N2	Skinny-128-384	128	96	128	96	48	128
	Romulus-N3	Skinny-128-256	128	96	128	96	24	128
Romulus-M	Romulus-M1	Skinny-128-384	128	128	128	128	56	128
	Romulus-M2	Skinny-128-384	128	96	128	96	48	128
	Romulus-M3	Skinny-128-256	128	96	128	96	24	128

B Instantiation of Romulus and Remus with Skinny

Endian. For both Romulus and Remus, we employ little endian for byte ordering: an n-bit string X is received as

$$X_7X_6\dots X_0 \,\|\, X_{15}X_{14}\dots X_8 \,\|\, \dots \,\|\, X_{n-1}X_{n-2}\dots X_{n-8},$$

where X_i denotes the $(i+1)$-st bit of X (for $i \in [\![n]\!]_0$). Therefore, when c is a multiple of 8 and X is a byte string, $\texttt{msb}_c(X)$ and $\texttt{lsb}_c(X)$ denote the last (rightmost) c bytes of X and the first (leftmost) c bytes of X, respectively. For example, $\texttt{lsb}_{16}(X) = (X_7X_6\dots X_0 \,\|\, X_{15}X_{14}\dots X_8)$ and $\texttt{msb}_8(X) = (X_{n-1}X_{n-2}\dots X_{n-8})$ with the above X. Since our specification is defined over byte strings, we only consider the above case for msb and lsb functions (i.e., the subscript c is always a multiple of 8). One can interpret $\texttt{lsb}_i$ and $\texttt{msb}_i$ as $\texttt{lmt}_i$ and $\texttt{rmt}_i$ in the specification of Section 3.

B.1 Instantiating Romulus with Skinny

We propose three versions of Romulus-N and three versions of Romulus-M when instantiated with Skinny. We provide in Table 9 the parameters for these variants.

B.1.1 The LFSR

We use LFSRs for counter. For positive integer c, $\texttt{lfsr}_c$ is a one-to-one mapping $\texttt{lfsr}_c : [\![2^c-1]\!]_0 \to \{0,1\}^c \backslash \{0^c\}$ defined as follows. For positive integer c, let $F_c(\mathtt{x})$ be the lexicographically-first polynomial among the the irreducible degree c polynomials of a minimum number of coefficients. Specifically $F_c(\mathtt{x})$ for $c \in \{56, 24\}$ are

$$F_{56}(\mathtt{x}) = \mathtt{x}^{56} + \mathtt{x}^7 + \mathtt{x}^4 + \mathtt{x}^2 + 1,$$
$$F_{24}(\mathtt{x}) = \mathtt{x}^{24} + \mathtt{x}^4 + \mathtt{x}^3 + \mathtt{x} + 1,$$

and

$$\texttt{lfsr}_c(D) = 2^D \bmod F_c(\mathtt{x}).$$

Note that we use $\texttt{lfsr}_c(D)$ as a block counter, so most of the time D changes incrementally with a step of 1, and this enables $\texttt{lfsr}_c(D)$ to generate a sequence of $2^c - 1$ pairwise-distinct values. From an implementation point of view, it should be implemented in the sequence form, $\mathtt{x}_{i+1} = 2 \cdot \mathtt{x}_i \bmod F_c(\mathtt{x})$.

Let $(z_{c-1} \,\|\, z_{c-2} \,\|\, \dots \,\|\, z_1 \,\|\, z_0)$ denote the state of c-bit LFSR. In our modes, these LFSRs are initialized to 1 mod $F_c(\mathtt{x})$, i.e., $(0^71 \,\|\, 0^{c-8})$, in little-endian format. Incrementation of

LFSRs is defined as follows: for $c = 56$,

$$\begin{aligned}
z_i &\leftarrow z_{i-1} \text{ for } i \in [\![56]\!]_0 \backslash \{7,4,2,0\},\\
z_7 &\leftarrow z_6 \oplus z_{55},\\
z_4 &\leftarrow z_3 \oplus z_{55},\\
z_2 &\leftarrow z_1 \oplus z_{55},\\
z_0 &\leftarrow z_{55}.
\end{aligned}$$

Similarly, for $c = 24$,

$$\begin{aligned}
z_i &\leftarrow z_{i-1} \text{ for } i \in [\![24]\!]_0 \backslash \{4,3,1,0\},\\
z_4 &\leftarrow z_3 \oplus z_{23},\\
z_3 &\leftarrow z_2 \oplus z_{23},\\
z_1 &\leftarrow z_0 \oplus z_{23},\\
z_0 &\leftarrow z_{23}.
\end{aligned}$$

Our LFSRs are also called *doubling* over GF(2^c) in the context of modes [Rog04a].

B.1.2 The Tweakey Encoding

Domain Separation. We will use a domain separation byte B to ensure appropriate independence between the tweakable block cipher calls and the various versions of Romulus. Let $B = (b_7 \| b_6 \| b_5 \| b_4 \| b_3 \| b_2 \| b_1 \| b_0)$ be the bitwise representation of this byte, where b_7 is the MSB and b_0 is the LSB (see also Figure 18). Then, we have the following:

- $b_7 b_6 b_5$ will specify the parameter sets. They are fixed to:
    - 000 for Romulus-N1
    - 001 for Romulus-M1
    - 010 for Romulus-N2
    - 011 for Romulus-M2
    - 100 for Romulus-N3
    - 101 for Romulus-M3

  Note that all nonce-respecting modes have $b_5 = 0$ and all nonce-misuse resistant modes have $b_5 = 1$.
- b_4 is set to 1 once we have handled the last block of data (AD and message chains are treated separately), to 0 otherwise.
- b_3 is set to 1 when we are performing the authentication phase of the operating mode (i.e., when no ciphertext data is produced), to 0 otherwise. In the special case where $b_5 = 1$ and $b_4 = 1$ (i.e., last block for the nonce-misuse mode), b_3 will instead denote if the number of message blocks is even ($b_5 = 1$ if that is the case, 0 otherwise).
- b_2 is set to 1 when we are handling a message block, to 0 otherwise. Note that in the case of the misuse-resistant modes, the message blocks will be used during authentication phase (in which case we will have $b_3 = 1$ and $b_2 = 1$). In the special case where $b_5 = 1$ and $b_4 = 1$ (i.e., last block for the nonce-misuse mode), b_3 will instead denote if the number of message blocks is even ($b_5 = 1$ if that is the case, 0 otherwise).
- b_1 is set to 1 when we are handling a padded AD block, to 0 otherwise.

- b_0 is set to 1 when we are handling a padded message block, to 0 otherwise.

The reader can refer to Table 10 to obtain the exact specifications of the domain separation values depending on the various cases.

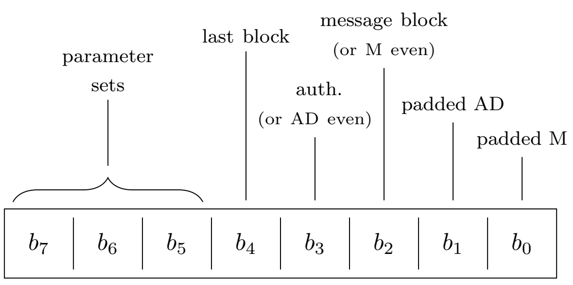

parameter sets
last block
auth. (or AD even)
message block (or M even)
padded AD
padded M
b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0

Figure 18: Domain separation when using the tweakable block cipher

Table 10: Domain separation byte B of Romulus. Bits b_7 and b_6 are to be set to the appropriate value according to the parameter sets.

	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	int(B)	case
Romulus-N	-	-	0	0	1	0	0	0	8	A main
	-	-	0	1	1	0	0	0	24	A last unpadded
	-	-	0	1	1	0	1	0	26	A last padded
	-	-	0	0	0	1	0	0	4	M main
	-	-	0	1	0	1	0	0	20	M last unpadded
	-	-	0	1	0	1	0	1	21	M last padded
Romulus-M	-	-	1	0	1	0	0	0	40	A main
	-	-	1	0	1	1	0	0	44	M auth main
	-	-	1	1	1	1	1	1	63	w: (even,even,padded,padded)
	-	-	1	1	1	1	1	0	62	w: (even,even,padded,unpadded)
	-	-	1	1	1	1	0	1	61	w: (even,even,unpadded,padded)
	-	-	1	1	1	1	0	0	60	w: (even,even,unpadded,unpadded)
	-	-	1	1	1	0	1	1	59	w: (even,odd,padded,padded)
	-	-	1	1	1	0	1	0	58	w: (even,odd,padded,unpadded)
	-	-	1	1	1	0	0	1	57	w: (even,odd,unpadded,padded)
	-	-	1	1	1	0	0	0	56	w: (even,odd,unpadded,unpadded)
	-	-	1	1	0	1	1	1	55	w: (odd,even,padded,padded)
	-	-	1	1	0	1	1	0	54	w: (odd,even,padded,unpadded)
	-	-	1	1	0	1	0	1	53	w: (odd,even,unpadded,padded)
	-	-	1	1	0	1	0	0	52	w: (odd,even,unpadded,unpadded)
	-	-	1	1	0	0	1	1	51	w: (odd,odd,padded,padded)
	-	-	1	1	0	0	1	0	50	w: (odd,odd,padded,unpadded)
	-	-	1	1	0	0	0	1	49	w: (odd,odd,unpadded,padded)
	-	-	1	1	0	0	0	0	48	w: (odd,odd,unpadded,unpadded)
	-	-	1	0	0	1	0	0	36	M enc main

Tweakey Encoding. We specify the following tweakey encoding functions for implementing TBC $\widetilde{E} : \mathcal{K} \times \overline{\mathcal{T}} \times \mathcal{M} \to \mathcal{M}$. The tweakey encoding is a function

$$\texttt{encode}_{m,t} : \mathcal{K} \times \overline{\mathcal{T}} \to \mathcal{K}_{\mathcal{T}},$$

where $\mathcal{K}_{\mathcal{T}} = \{0,1\}^m$ is the tweakey space. As defined earlier, $\overline{\mathcal{T}} = \mathcal{T} \times \mathcal{B} \times \mathcal{D}$, $\mathcal{K} = \{0,1\}^k$ and $\mathcal{T} = \{0,1\}^t$, $\mathcal{D} = [\![2^d - 1]\!]_0$, $\mathcal{B} = [\![256]\!]_0$.

Table 11: Members of Remus.

Family	Name	E	`ICmode`	k	nl	n	d	τ
Remus-N	Remus-N1	Skinny-128/128	ICE1	128	128	128	128	128
	Remus-N2	Skinny-128/128	ICE2	128	128	128	128	128
	Remus-N3	Skinny-64/128	ICE3	128	96	64	120	64
Remus-M	Remus-M1	Skinny-128/128	ICE1	128	128	128	128	128
	Remus-M2	Skinny-128/128	ICE2	128	128	128	128	128

- Case $(m, t) = (384, 128)$: this variant is used for Romulus-N1 and Romulus-M1. The `encode` function is defined as follows:

$$\texttt{encode}_{384,128}(K, T, B, D) = \texttt{lfsr}_{56}(D) \parallel B \parallel 0^{64} \parallel T \parallel K$$

- Case $(m, t) = (384, 96)$: this variant is used for Romulus-N2 and Romulus-M2. The `encode` function is defined as follows:

$$\texttt{encode}_{384,96}(K, T, B, D) = \texttt{lfsr}_{24}(D_1) \parallel B \parallel T \parallel K \parallel \texttt{lfsr}_{24}(D_2) \parallel 0^{104},$$

  where $D_1, D_2 \in \mathcal{D}_s$ with $\mathcal{D}_s = [\![2^{24} - 1]\!]$. The set $\mathcal{D}$ is defined as $\mathcal{D} = \mathcal{D}_s \times \mathcal{D}_s$, and the components are determined from D as $D_1 = (D/(2^{24} - 1)) + 1$ and $D_2 = (D \bmod (2^{24} - 1)) + 1$. For the first $2^{24} - 1$ cycles starting from $D = 0$ (but note that $D = 1$ is the initial value in our scheme and $D = 0$ is not used), D_1 is fixed to 1 and D_2 takes all integers of $[\![2^{24} - 1]\!]$. For the next $2^{24} - 1$ cycles, D_1 is fixed to 2 and D_2 takes all values of $[\![2^{24} - 1]\!]$ again, and so on. We stress that the counter cycle is $(2^{24} - 1)^2$ which is slightly smaller than the original range of D. One can also interpret D as a two-dimensional vector: for example, if $D_1 = 0^{23}1$ and $D_2 = 0^{20}1^4$, then $D = (0^{23}1 \parallel 0^{20}1^4)$. In this case, the initial value of D is $[0^7 10^{16}, 0^7 10^{16}]$, in little-endian format.

- Case $(m, t) = (256, 96)$: this variant is used for Romulus-N3 and Romulus-M3. The `encode` function is defined as follows:

$$\texttt{encode}_{256,96}(K, T, B, D) = \texttt{lfsr}_{24}(D) \parallel B \parallel T \parallel K$$

For plaintext $M \in \{0,1\}^n$ and tweak $\overline{\mathcal{T}} = (T, B, D) \in \mathcal{T} \times \mathcal{B} \times \mathcal{D}$, $\widetilde{E}_K^{(T,B,D)}(M)$ denotes encryption of M with m-bit tweakey state $\texttt{encode}_{m,t}(K, T, B, D)$. Tweakey encode is always implicitly applied, hence the counter D is never arithmetic in the tweakey state. To avoid confusion, we may write $\overline{D}$ (in particular when it appears in a part of tweak) in order to emphasize that this is indeed an LFSR counter. One can interpret $\overline{D}$ as a state of the LFSR after D clocks (but in that case it is a part of the tweakey state and *not* a part of input of `encode`).

B.2 Instantiating Remus with Skinny

We propose three versions of Remus-N and two versions of Remus-M when instantiated with Skinny. We provide in Table 11 the parameters for these variants.

Doubling over a Finite Field. For any positive integer c, we assume GF(2^c) is defined over the lexicographically-first polynomial among the irreducible degree c polynomials of a minimum number of coefficients. We use two fields: $\mathrm{GF}(2^c)$ for $c \in \{128, 120\}$. The primitive polynomials are:

$$\begin{aligned}&\mathtt{x}^{128} + \mathtt{x}^7 + \mathtt{x}^2 + \mathtt{x} + 1 \text{ for } c = 128,\\&\mathtt{x}^{120} + \mathtt{x}^4 + \mathtt{x}^3 + \mathtt{x} + 1 \text{ for } c = 120.\end{aligned}$$

Let $Z = (z_{c-1}z_{c-2}\dots z_1z_0)$ for $z_i \in \{0,1\}$, $i \in [\![c]\!]_0$ be an element of $\mathrm{GF}(2^c)$. A multiplication of Z by the generator (polynomial $\mathtt{x}$) is called *doubling* and written as $2Z$ [Rog04a]. An i-times doubling of Z is written as 2^iZ, and is efficiently computed from $2^{i-1}Z$ (see below). Here, $2^0Z = Z$ for any Z. When $Z = 0^n$, i.e., zero entity in the field, then $2^iZ = 0^n$ for any $i \geqslant 0$.

To avoid confusion, we may write $\overline{D}$ (in particular when it appears in a part of tweak) in order to emphasize that this is indeed a doubling-based counter, i.e., 2^DX for some key-dependent variable X. One can interpret $\overline{D}$ as 2^D (but in that case it is a part of tweakey state or a coefficient of mask, and *not* a part of input of ICE).

On bit-level, doubling $Z \to 2Z$ over $\mathrm{GF}(2^c)$ for $c = 128$ is defined as

$$\begin{aligned}&z_i \leftarrow z_{i-1} \text{ for } i \in [\![128]\!]_0\backslash\{7,2,1,0\},\\&z_7 \leftarrow z_6 \oplus z_{127},\\&z_2 \leftarrow z_1 \oplus z_{127},\\&z_1 \leftarrow z_0 \oplus z_{127},\\&z_0 \leftarrow z_{127}.\end{aligned}$$

Similarly, for $\mathrm{GF}(2^{120})$, we have

$$\begin{aligned}&z_i \leftarrow z_{i-1} \text{ for } i \in [\![120]\!]_0\backslash\{4,3,1,0\},\\&z_4 \leftarrow z_3 \oplus z_{119},\\&z_3 \leftarrow z_2 \oplus z_{119},\\&z_1 \leftarrow z_0 \oplus z_{119},\\&z_0 \leftarrow z_{119}.\end{aligned}$$

B.2.1 The TBC ICE

The three variants of ICE are defined as follows. See Section 3.3 for reference.

- ICE1: $n = 128$, $nl = 128$, $d = 128$, $k = 128$ and it uses Skinny-128/128 as its building block E.
- ICE2: $n = 128$, $nl = 128$, $d = 128$, $k = 128$ and it uses Skinny-128/128 as its building block E.
- ICE3: $n = 64$, $nl = 96$, $d = 120$, $k = 128$ and it uses Skinny-64/128 as its building block E.

B.2.2 Block Counters and Domain Separation

Domain Separation. We will use a domain separation byte B to ensure appropriate independence between the tweakable block cipher calls and the various versions of Remus. Let $B = (b_7\|b_6\|b_5\|b_4\|b_3\|b_2\|b_1\|b_0)$ be the bitwise representation of this byte, where b_7 is the MSB and b_0 is the LSB (see also Figure 19). Then, we have the following:

- $b_7b_6b_5$ will specify the parameter sets. They are fixed to:
  - 000 for Remus-N1
  - 001 for Remus-M1
  - 010 for Remus-N2
  - 011 for Remus-M2
  - 100 for Remus-N3

  Note that all nonce-respecting modes have $b_5 = 0$ and all nonce-misuse resistant modes have $b_5 = 1$.
- b_4 is set to 0.
- b_3 is set to 1 once we have handled the last block of data (AD and message chains are treated separately), to 0 otherwise.
- b_2 is set to 1 when we are performing the authentication phase of the operating mode (i.e., when no ciphertext data is produced), to 0 otherwise. In the special case where $b_5 = 1$ and $b_4 = 1$ (i.e., last block for the nonce-misuse mode), b_3 will instead denote if the number of message blocks is even ($b_5 = 1$ if that is the case, 0 otherwise).
- b_1 is set to 1 when we are handling a message block, to 0 otherwise. Note that in the case of the misuse-resistant modes, the message blocks will be used during authentication phase (in which case we will have $b_3 = 1$ and $b_2 = 1$). In the special case where $b_5 = 1$ and $b_4 = 1$ (i.e., last block for the nonce-misuse mode), b_3 will instead denote if the number of message blocks is even ($b_5 = 1$ if that is the case, 0 otherwise).
- b_0 is set to 1 when we are handling a padded block (associated data or message), to 0 otherwise.

The reader can refer to Table 12 to obtain the exact specifications of the domain separation values depending on the various cases.

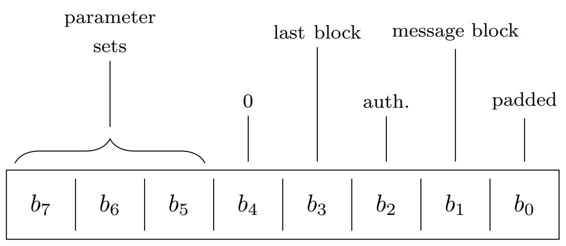


Figure 19: Domain separation when using the tweakable block cipher

Table 12: Domain separation byte B of Remus. Bits b_7 and b_6 are to be set to the appropriate value according to the parameter sets.

	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	int(B)	case
Remus-N	-	-	0	0	0	1	0	0	4	A main
	-	-	0	0	1	1	0	0	12	A last unpadded
	-	-	0	0	1	1	0	1	13	A last padded
	-	-	0	0	0	0	1	0	2	M main
	-	-	0	0	1	0	1	0	10	M last unpadded
	-	-	0	0	1	0	1	1	11	M last padded
Remus-M	-	-	1	0	0	1	0	0	36	A main
	-	-	1	0	1	1	0	0	44	A last unpadded
	-	-	1	0	1	1	0	1	45	A last padded
	-	-	1	0	0	1	1	0	38	M auth main
	-	-	1	0	1	1	1	0	46	M auth last unpadded
	-	-	1	0	1	1	1	1	47	M auth last padded
	-	-	1	0	0	0	1	0	34	M enc main