SUNDAE: Small Universal Deterministic Authenticated Encryption for the Internet of Things

Subhadeep Banik[1], Andrey Bogdanov[2], Atul Luykx[3], Elmar Tischhauser[2]

[1] LASEC, École Polytechnique Fédérale de Lausanne, Switzerland
subhadeep.banik@epfl.ch
[2] Technical University of Denmark, Denmark
{anbog,ewti}@dtu.dk
[3] Visa Research, USA
aluykx@visa.com

Abstract. Lightweight cryptography was developed in response to the increasing need to secure devices for the Internet of Things. After significant research effort, many new block ciphers have been designed targeting lightweight settings, optimizing efficiency metrics which conventional block ciphers did not. However, block ciphers must be used in modes of operation to achieve more advanced security goals such as data confidentiality and authenticity, a research area given relatively little attention in the lightweight setting. We introduce a new authenticated encryption (AE) mode of operation, SUNDAE, specially targeted for constrained environments. SUNDAE is smaller than other known lightweight modes in implementation area, such as CLOC, JAMBU, and COFB, however unlike these modes, SUNDAE is designed as a deterministic authenticated encryption (DAE) scheme, meaning it provides maximal security in settings where proper randomness is hard to generate, or secure storage must be minimized due to expense. Unlike other DAE schemes, such as GCM-SIV, SUNDAE can be implemented efficiently on both constrained devices, as well as the servers communicating with those devices. We prove SUNDAE secure relative to its underlying block cipher, and provide an extensive implementation study, with results in both software and hardware, demonstrating that SUNDAE offers improved compactness and power consumption in hardware compared to other lightweight AE modes, while simultaneously offering comparable performance to GCM-SIV on parallel high-end platforms.

Keywords: lightweight · block cipher · mode of operation · deterministic authenticated encryption · nonce misuse resistance

1 Introduction

As computing on increasingly small devices becomes widespread, enabling security — and in particular cryptography — in such constrained environments becomes critical. Recognizing the fact that cryptographic algorithms optimized for high-performance computing are not necessarily optimal for constrained environments, various research and standardization efforts have set out to explore *lightweight* cryptography, such as ISO/IEC 29192, CRYPTREC, the CAESAR competition [CAE16], and NIST's lightweight project [nis17].

Lightweight block cipher design is one of the most mature research areas, with constructions going back to 2007, optimizing for a variety of efficiency goals such as latency [BJK+16], area [CDK09], and energy [BBI+15]. Although optimizing block cipher design is an important first step, block ciphers on their own are only building blocks, and should be used in

IACR Transactions on Symmetric Cryptology ISSN 2519-173X, Vol. 2018, No. 3, pp. 1–35
DOI:10.13154/tosc.v2018.i3.1-35

modes of operation to achieve security. In particular, ensuring data confidentiality and authenticity is done using an *authenticated encryption* (AE) mode of operation.

Even if a block cipher is ideally suited to a given environment when considered in isolation, it could be used in an AE mode of operation which erases many of the block cipher's benefits. In fact, AE modes are often not designed to account for the different requirements imposed by lightweight settings. The mode might require two separate, independent keys, as with SIV [RS06], a state size of at least thrice that of the underlying block size, as with COPA [ABL+13], or multiple initial block cipher calls before it can start processing data, like in EAX [BRW04].

Exceptions include the following AE modes CLOC [IMGM14], JAMBU [WH16], and COFB [CIMN17a], which try to reduce state size and number of block cipher calls to optimize for short messages. However, the challenges imposed by constrained environments are not limited to efficiency constraints, as fundamental security assumptions might be difficult to guarantee as well. For example, devices might lack proper randomness sources, or have limited secure storage to maintain state, in which case they might not be able to generate the nonces necessary to ensure that modes such as CLOC, JAMBU, and COFB maintain security. In such cases, algorithms which provide more robust security are better, such as nonce-misuse resistant AE [RS06], as they do not fail outright in the wrong conditions.

An efficient nonce-misuse resistant dedicated instantiation of SIV called GCM-SIV was proposed by Gueron and Lindell at CCS 2015 [GL15]. While it attains very competitive performance in software on recent Intel architectures, it requires full multiplications in $GF(2^{128})$, which makes the scheme unattractive in hardware and on resource-constraint platforms. The importance of good implementation characteristics on all platforms was already pointed out in [MM12]: the same cryptographic algorithms used on the small devices of the Internet of Things also have to be employed on the servers that are communicating with them. Crucially, however, the few designs explicitly aiming at being simultaneously efficient on lightweight as well as high-performance platforms such as [BMR+13, LPTY16] do not provide nonce-misuse resistant authenticated encryption. In this paper, we aim to address this gap.

1.1 Contributions

We introduce an AE mode of operation, SUNDAE, which

1. competes with CLOC and JAMBU in number of block cipher calls for short messages,
2. improves over those algorithms and COFB in terms of state size,
3. provides maximal robustness to a lack of proper randomness or secure state, and
4. simultaneously offers good implementation characteristics on lightweight and high-performance platforms.

SUNDAE is designed to be a *deterministic authenticated encryption* mode [RS06], which means that as long as its input is unique, it maintains both data confidentiality and authenticity. If inputs are repeated, then only that fact is leaked.

SUNDAE processes inputs of the form (A, M), where A is associated data which need not be encrypted, and M is plaintext data to be encrypted. If M is empty, then SUNDAE becomes a MAC algorithm. If needed, nonces are included as the first x bits of associated data, where x is a parameter fixing the nonce's length per key.

SUNDAE's structure is based on SIV [RS06], however it is optimized for lightweight settings: it uses one key, consists of a cascade of block cipher calls, and its only additional operations consist of XOR and multiplication by fixed constants. The use of efficienct intermediate functions is inspired by GCBC [Nan09]. Using an n-bit block cipher, aside from storage for the key, CLOC requires $2n$-bit state, JAMBU $1.5n$-bit state, and COFB $1.5n$-bit state, whereas SUNDAE only uses an n-bit state.

SUNDAE's performance is fundamentally limited by the fact that it requires two block cipher calls per data block, hence SUNDAE works best for communication which consists of short messages. For a message consisting of one block of nonce, associated data, and plaintext, COFB uses 3 block cipher calls, CLOC requires 4, JAMBU 5, and SUNDAE 5 as well (which can be reduced to 4 if one block cipher call can be precomputed). However, SUNDAE's strength lies in settings where communication outweighs computational costs: if the combination of associated data and plaintext is never repeated, the nonce is no longer needed, and communication or synchronization costs are reduced, in addition to reducing the block cipher calls to 4.

SUNDAE is inherently serial, and although the client side is important, it is not everything, especially given GCM-SIV's excellent performance using AES-NI on Haswell and Skylake. Even though parallel modes inherently profit most from modern parallel architectures, the Comb scheduling technique proposed in [BLT15] can mitigate this issue even for serial modes, at least on the server side. Therefore, we can afford to deploy a serial approach to design a novel mode of operation.

1.2 Related Work

Aside from lightweight block ciphers, many other primitives have been optimized for the lightweight setting, such as stream ciphers [CP08] and hash functions [BKL+13, GPP11], as well as dedicated designs achieving authenticated encryption, such as Grain-128a [ÅHJM11]. Furthermore, permutation-based cryptography provides an approach to designing authenticated encryption with better trade-offs suited to lightweight settings [DEMS14, ABB+14a, BDP+14a, BDP+14b, AJN14, BDPV11, ABB+14b]. Applications which have the flexibility to choose the underlying primitive will often find the better choice in using permutation-based cryptography. However, there are settings where for legacy reasons one is restricted to using block ciphers — our focus is on designing a scheme for such settings.

Nonce-misuse resistance has been studied extensively over the past years. Initially introduced by Rogaway and Shrimpton [RS06], constructions include HBS [IY09b], BTM [IY09a], SCT [PS16], and GCM-SIV [GL15]. Böck, Zauner, Devlin, Somorovsky, and Jovanovic [BZD+16] investigate the applicability of nonce-misusing attacks in TLS by searching for servers which repeat nonces with GCM. Security definitions describing weaker levels of nonce misuse have been explored as well [FFL12, HRRV15, ABL+13, ADL17].

When run with empty plaintext data, SUNDAE looks similar to MAC algorithms such as GCBC [Nan09] and CBCR [ZWZW11].

2 Notation

Unless specified otherwise, all sets are finite. If X is a set, then X^n is the set of length-n sequences of X, $\mathsf{X}^{\leq q}$ the set of sequences of X of length not greater than q including the empty sequence, and X^* the set of finite-length sequences of X. If $X \in \mathsf{X}^*$, then $|X|$ is its length. Given $X, Y \in \mathsf{X}^*$, concatenation of X and Y is denoted $X\|Y$, or simply XY when no confusion arises.

The notation $x \mapsto y$ is used to denote a function which maps the symbol x on the left to the symbol y on the right. If f is a function with domain $\mathsf{X} \times \mathsf{Y}$, then we write $f(X, Y)$ and $f_X(Y)$ interchangeably, and use the notation f_X to denote the function obtained by fixing the first input of f to X.

Throughout the paper, n denotes *block size*. The set of *blocks* is $\{0,1\}^{\leq n}$, and $\mathsf{B} := \{0,1\}^n$ denotes the subset of *complete* blocks, with all other blocks called *incomplete*. The element $0^n \in \mathsf{B}$ denotes the complete block consisting of only zeroes, and the function $\mathsf{pad} : \{0,1\}^{\leq n} \to \mathsf{B}$ pads an incomplete block X with a 1 followed by $n - |X| - 1$ zeroes,

and leaves complete blocks as-is:

$$\mathsf{pad}(X) = \begin{cases} X \parallel 10^{n-|X|-1} & \text{if } |X| < n \\ X & \text{otherwise}\,. \end{cases} \tag{1}$$

The empty string is denoted ε. Given two equal-length elements $X, Y \in \{0,1\}^*$, $X \oplus Y$ denotes their bitwise XOR. If $X \in \{0,1\}^*$, then $\lfloor X \rfloor_m$ denotes truncating X to the m most significant bits of X. Splitting a non-empty string X into blocks is done by computing its *block length* ℓ, which is the smallest integer greater than or equal to $|X|/n$, and processing X as

$$X[1]X[2]\cdots X[\ell-1]X[\ell] \xleftarrow{n} X \tag{2}$$

where $|X[i]| = n$ for $1 \le i < \ell$, and $0 < |X[\ell]| \le n$.

The set of complete blocks can be viewed as a finite field by mapping strings to polynomials over finite fields. For a positive divisor i of n, we map the bits $b_0, \ldots, b_{n-1}$ to i elements of $GF(2)[\mathtt{x}]/(m)$ for a fixed irreducible polynomial $m(\mathtt{x})$ of degree n/i over $GF(2)$: $a_0 = b_0 + b_i\mathtt{x} + \cdots + b_{n-i}\mathtt{x}^{n/i-1}, a_1 = b_1 + b_{i+1}\mathtt{x} + \cdots + b_{n-i+1}\mathtt{x}^{n/i-1}, \ldots, a_{i-1} = b_{i-1} + b_{2i-1}\mathtt{x} + \cdots + b_{n-1}\mathtt{x}^{n/i-1}$. Given such a mapping and $X \in \mathsf{B}$, we let $2 \times X$ and $4 \times X$ denote multiplication by $\mathtt{x}$ and $\mathtt{x}^2$ of all $a_0, \ldots, a_{i-1}$ in their polynomial basis, respectively.

Concrete instantiations for $n = 64, 128$ are proposed in Sect. 5.2, optimized for block ciphers with 4-bit or 8-bit S-boxes, respectively.

The function $\mathsf{E} : \mathsf{K} \times \mathsf{B} \to \mathsf{B}$ denotes a block cipher, with K the set of keys. The expression $a\ ?\ b : c$ evaluates to b if a is true and c otherwise.

3 Specification

SUNDAE consists of an encryption algorithm enc and a decryption algorithm dec. It is parametrized by a block cipher $\mathsf{E} : \mathsf{K} \times \mathsf{B} \to \mathsf{B}$, which fixes a key set K and block size n, and a representation of B as a finite field. The encryption algorithm enc takes as input a key $K \in \mathsf{K}$, associated data $A \in \{0,1\}^*$, and a message $M \in \{0,1\}^*$. It outputs a ciphertext $C \in \{0,1\}^{n+|M|}$, where the first n bits of the ciphertext are interpreted as a *tag*. The decryption algorithm dec takes as input a key $K \in \mathsf{K}$, associated data $A \in \{0,1\}^*$, and a ciphertext $C \in \{0,1\}^n \times \{0,1\}^*$, and outputs $M \in \{0,1\}^{|C|-n}$, or the error symbol $\perp$ if verification is not successful. The encryption and decryption algorithms are such that for all $K \in \mathsf{K}$, $A \in \{0,1\}^*$, $M \in \{0,1\}^*$, with $|A| + |M| > 0$,

$$\mathsf{dec}_K\big(A, \mathsf{enc}_K(A, M)\big) = M\,. \tag{3}$$

The key $K \in \mathsf{K}$ must be generated as specified by the underlying block cipher E, which usually involves choosing K uniformly at random from K. After fixing a key, uniqueness should be guaranteed of each pair (A, M) of associated data and message input; associated data can be repeated if the message is changed, and message input may be repeated if the associated data is changed. Caution must be taken so that intermediate values used during encryption and decryption are not leaked. In particular, unverified plaintext from the decryption algorithm should not be released [ABL+14]. Finally, proper operation of SUNDAE requires changing keys well before the bound from Thm. 1 becomes void.

Alg. 1 and Alg. 2 provide pseudocode for enc and dec respectively, and Fig. 1 gives a diagram of encryption. All block cipher calls are performed with a fixed key K. Both encryption and decryption algorithms only use the "forward" block cipher E_K, hence the block cipher inverse is not needed.

Algorithm 1: $\mathsf{enc}_K(A, M)$

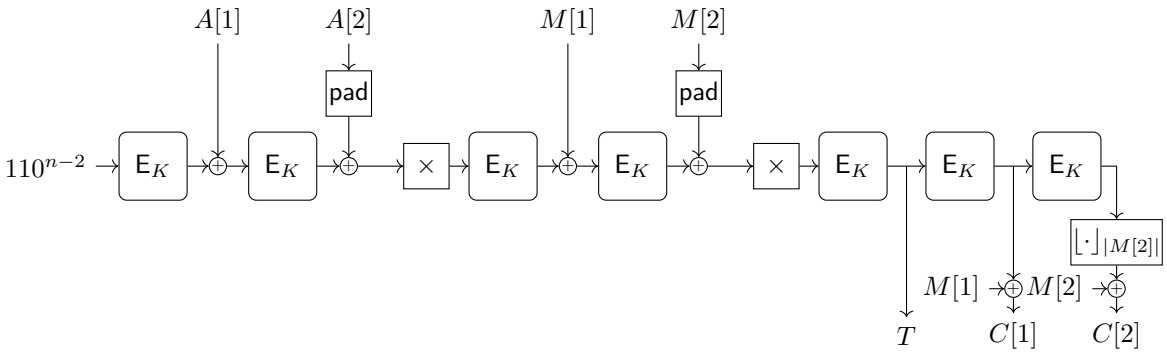
Input: $K \in \mathsf{K}$, $A \in \{0,1\}^*$, $M \in \{0,1\}^*$
Output: $C \in \{0,1\}^{n+|M|}$

$b_1 \leftarrow |A| > 0 \ ? \ 1 : 0$
$b_2 \leftarrow |M| > 0 \ ? \ 1 : 0$
$V \leftarrow \mathsf{E}_K\big(b_1 \| b_2 \| 0^{n-2}\big)$
$T \leftarrow V$ `// Initial tag`
if $|A| > 0$ **then**
  $A[1]A[2]\cdots A[\ell_A] \xleftarrow{n} A$
  for $i = 1$ **to** $\ell_A - 1$ **do**
    $V \leftarrow \mathsf{E}_K\big(V \oplus A[i]\big)$
  end
  $X \leftarrow |A[\ell_A]| < n \ ? \ 2 : 4$
  $V \leftarrow \mathsf{E}_K\Big(X \times \big(V \oplus \mathsf{pad}(A[\ell_A])\big)\Big)$
  $T \leftarrow V$
end
if $|M| > 0$ **then**
  $M[1]M[2]\cdots M[\ell_M] \xleftarrow{n} M$
  for $i = 1$ **to** $\ell_M - 1$ **do**
    $V \leftarrow \mathsf{E}_K\big(V \oplus M[i]\big)$
  end
  $X \leftarrow |M[\ell_M]| < n \ ? \ 2 : 4$
  $V \leftarrow \mathsf{E}_K\Big(X \times \big(V \oplus \mathsf{pad}(M[\ell_M])\big)\Big)$
  $T \leftarrow V$
  for $i = 1$ **to** ℓ_M **do**
    $V \leftarrow \mathsf{E}_K\big(V\big)$
    $C[i] \leftarrow \lfloor V \rfloor_{|M[i]|} \oplus M[i]$
  end
  return $TC[1]\cdots C[\ell_M]$
end
return T

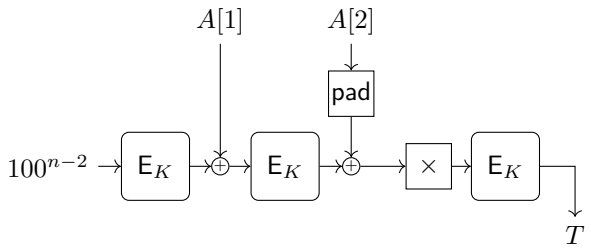
Algorithm 2: $\mathsf{dec}_K(A, C)$

Input: $K \in \mathsf{K}$, $A \in \{0,1\}^*$, $C \in \{0,1\}^n \times \{0,1\}^*$
Output: $\perp$ or $M \in \{0,1\}^{|C|-n}$

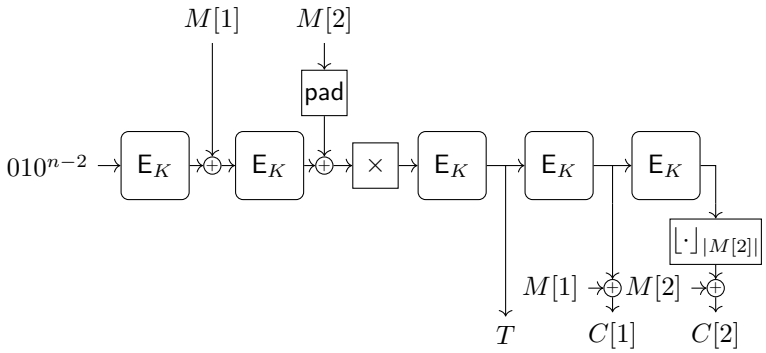
$C[1]C[2]\cdots C[\ell] \xleftarrow{n} C$
$V \leftarrow C[1]$
for $i = 2$ **to** ℓ **do**
  $V \leftarrow E_K\big(V\big)$
  $M[i-1] \leftarrow \lfloor V \rfloor_{|M[i]|} \oplus C[i]$
end
$M \leftarrow \ell > 1 \ ? \ M[1]M[2]\cdots M[\ell-1] : \varepsilon$
$T \leftarrow \lfloor \mathsf{enc}_K(A, M) \rfloor_n$
if $T \neq C[1]$ **then**
  return $\perp$
return M



(a) SUNDAE encryption with associated and plaintext data. The box below the rightmost block cipher call represents truncation.



(b) SUNDAE without plaintext data, meaning only a tag is produced, like a MAC.



(c) SUNDAE encryption with only plaintext data.

Figure 1: Diagrams of SUNDAE encryption and authentication. The initial block cipher call changes depending upon the presence of associated and plaintext data. The multiplication × by 2 or 4 and depends on the length of the last blocks.

4 Security Analysis

4.1 Intuition and Proof Overview

SUNDAE is analyzed as a *deterministic authenticated encryption (DAE)* algorithm [RS06], and therefore must achieve authenticity, and confidentiality up to repetition of inputs. Although our formal analysis considers confidentiality and authenticity simultaneously, here we give an intuitive explanation which considers the two goals separately.

SUNDAE generally follows the MAC-then-encrypt paradigm, much like SIV [RS06], since SUNDAE processes associated data and plaintext first with a MAC algorithm, and then uses the MAC algorithm output as the "IV" input to the stream cipher OFB [Nat80].

However, the use of a single key for both the MAC algorithm and stream cipher means that SIV's analysis does not carry over. Furthermore, although SUNDAE exhibits similarities with GCBC [Nan09] and OFB, the analyses of those schemes have limited applicability to SUNDAE as the combination of deterministic encryption and authentication introduces complications which do not arise when trying to achieve the goals separately. Therefore a proof of SUNDAE requires new arguments, albeit using techniques from throughout the literature.

Our formal argument starts with the simplifying step of applying a PRP-PRF switch to analyze SUNDAE with uniform random function ρ. Although this limits our analysis to the birthday bound, SUNDAE's security will anyway be limited by birthday bound attacks, for the same reasons the CBC and OFB encryption modes are.

Confidentiality After the PRP-PRF switch, confidentiality can be argued as follows. Since plaintext is encrypted into ciphertext using the stream cipher OFB, confidentiality is maintained if the stream cipher output looks uniformly random to the adversary. In the proof we end up setting aside the fact that plaintext is XORed with the stream cipher output to produce ciphertext since we are considering chosen-plaintext adversaries; the resulting simplified construction is denoted enc-stream in the proof.

OFB maintains security if its "IV" is unpredictable to the adversary. In the case of SUNDAE, the IV corresponds to the tag, hence intuitively, confidentiality will be maintained if the tag is unpredictable. Unlike OFB, we need to take into account the MAC algorithm which produces the tag. Complications arise due to the fact that associated data and plaintext data are processed similarly, with the main method of domain separation being the intermediate functions.

A large part of the formal argument is proving that SUNDAE's domain separation works. To do so, we calculate the probability that any two ρ-inputs collide in a meaningful way — a meaningless ρ-collision would be one where the adversary keeps the prefix of two different queries the same, resulting in the same ρ-input since SUNDAE is deterministic. As long as there are no meaningful collisions, it is easy to argue SUNDAE's confidentiality. Many of the details in the calculation of the ρ-collision bound have little to do with SUNDAE or with the specific intermediate functions that we choose, hence we abstract away details of the argument and prove a more general result in Sect. 4.10.

To connect SUNDAE with the abstract analysis of Sect. 4.10, we recast SUNDAE into different notation in Sect. 4.5 so that the intermediate functions become explicit. This way SUNDAE can be viewed as a cascade of ρ-queries, alternated with intermediate function calls. The sequence of intermediate function calls is denoted $I(A, M)$, which makes explicit the fact that they only depend on the associated data and plaintext input. Then, following Patarin's method, we focus on analyzing transcripts of interactions between adversaries and SUNDAE, thereby fixing adversarial input, and as a result intermediate functions. Each transcript then gets converted into a graph in Sect. 4.8.3, which characterizes the relationship between all the ρ-inputs: each node of the graph represents an intermediate function, and if you follow the graph from the "root" node to a leaf node, while applying ρ while going from node to node, you will have executed a call to SUNDAE.

Once the connection between transcripts and graphs has been established, Sect. 4.8.4 describes the types of collisions that can occur, with the important ones being "structural" and "accidental". A structural collision is one which would happen if $I(A, M)$ were poorly designed, by, for example, using the same intermediate functions for two unrelated inputs. Analysis of accidental collisions is done in Sect. 4.10.

Authenticity Consider an adversary which somehow produces a forgery (C, T). This means it found a tag T such that the output of the MAC algorithm during the (C, T)-decryption call equals T. In particular, intuitively, it would have had to have found a

pre-image or second pre-image of the underlying MAC algorithm, since, by definition, C was never output by a previous encryption query (otherwise it would not be a valid forgery). The bulk of the formal argument involves showing that it is in fact difficult for the adversary to produce such an event.

We introduce an intermediate construction to arrive at our conclusion. First, the decryption algorithm of SUNDAE is rewritten in terms of enc-stream and stream, the latter essentially describing OFB mode. After removing the plaintext XOR, we end up with the oracles (enc-stream, dec-stream). Looking more closely at dec-stream, one sees that its internal stream call could be recreated with the enc-stream output that is available to the adversary, since any dec-stream call that uses a tag input which is equal to some previous enc-stream output, will make the exact same ρ-calls as that previous enc-stream query. It is only when a dec-stream call is made which uses tag input which is unrelated to all previous enc-stream output, or when a sufficiently long dec-stream call is made that new ρ-queries are made. Using this knowledge, we introduce the intermediate construction dec-stream*, which tries to recreate dec-stream as best as possible using newly generated uniform random values if necessary to recreate missing ρ-calls.

The argument on the difficultly to find pre-images and second pre-images is easy to reason about with dec-stream*, as shown in Sect. 4.7. The rest of the authenticity proof focuses on bounding the distance between (enc-stream, dec-stream) and the intermediate construction with dec-stream*. Then, as explained above for confidentiality, as long as no ρ-collision occurs, the adversary will not be able to distinguish SUNDAE from the intermediate world which uses dec-stream*. The general result from Sect. 4.10 is then re-used.

4.2 Security Definitions, and Statements

For the security definitions we will need the following concepts. Oracles and adversaries are probabilistic algorithms. Given two sequences of oracles $(f_1, \ldots, f_\mu)$ and $(g_1, \ldots, g_\mu)$, we denote the advantage of an adversary $\mathbf{A}$ in distinguishing $(f_1, \ldots, f_\mu)$ from $(g_1, \ldots, g_\mu)$ by

$$\underset{\mathbf{A}}{\Delta}(f_1, \ldots, f_\mu\,;\, g_1, \ldots, g_\mu) := \left|\mathbf{P}\left[\mathbf{A}^{f_1,\ldots,f_\mu} \to 1\right] - \mathbf{P}\left[\mathbf{A}^{g_1,\ldots,g_\mu} \to 1\right]\right|, \tag{4}$$

where the notation $\mathbf{A}^{O_1,\ldots,O_\mu} \to 1$ indicates that $\mathbf{A}$ outputs 1 when interacting with oracles $O_1, \ldots, O_\mu$.

A *uniformly distributed random function (URF)* over X is a random variable uniformly distributed over the set of all functions on X. Let $\$_{i,j,k}$ represent a family of independent URFs from $\{0,1\}^i \times \{0,1\}^j$ to $\{0,1\}^k$ for $i, j, k \geq 0$, then define $\$: \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ to be

$$\$(A, M) := \$_{|A|,|M|,n+|M|}(A, M)\,. \tag{5}$$

Definition 1 (DAE Security)**.** Adversary $\mathbf{A}$'s DAE advantage against SUNDAE is defined as

$$\mathsf{DAE}(\mathbf{A}) := \underset{\mathbf{A}}{\Delta}(\mathsf{enc}_k, \mathsf{dec}_K\,;\, \$, \perp)\,, \tag{6}$$

where K is chosen uniformly at random from K and $\perp$ is an oracle that always outputs $\perp$. Letting (O_1, O_2) denote the oracles $\mathbf{A}$ interacts with, the adversary may not query $O_2(A, C)$ if previously a query $O_1(A, M)$ with output C was made.

We follow the concrete security paradigm [BDJR97] by explicitly describing SUNDAE's security in terms of adversarial resources. An adversary's *associated data block length cost* is the sum of the block lengths of the associated data that it queries to either SUNDAE's encryption or decryption algorithms. Plaintext and ciphertext block length costs are defined similarly, with an adversary's *total block length cost* defined as the sum of its

associated data, plaintext, and ciphertext block length costs. If σ_A, σ_P, and σ_C denote $\mathbf{A}$'s associated data, plaintext, and ciphertext block length costs, then $\mathbf{A}$ makes at most

$$N_{\mathsf{E}} := 4 + \sigma_A + 2\sigma_P + 2\sigma_C \tag{7}$$

block cipher calls indirectly via SUNDAE.

SUNDAE is a mode of operation, hence its security relies on the quality of its underlying block cipher, defined as follows.

Definition 2 (PRP Advantage)**.** Let $\mathsf{E} : \mathsf{K} \times \mathsf{B} \to \mathsf{B}$ be a block cipher. Then adversary $\mathbf{A}$'s PRP-advantage against E is

$$\mathsf{PRP}_{\mathsf{E}}(\mathbf{A}) := \underset{\mathbf{A}}{\Delta}(\mathsf{E}_K\,;\,\pi)\ , \tag{8}$$

where K is chosen uniformly at random from K, and π is chosen uniformly at random from the set of all permutations on X.

Any adversary $\mathbf{A}$ against SUNDAE can be converted into an adversary $\mathbf{A}_{\mathsf{E}}$ against the block cipher as follows: adversary $\mathbf{A}_{\mathsf{E}}$ runs $\mathbf{A}$, and each time $\mathbf{A}$ makes a query to SUNDAE, $\mathbf{A}_{\mathsf{E}}$ recreates SUNDAE encryption or decryption with its own oracle, either E_K or π, according to SUNDAE's definition.

Theorem 1. *Let $\mathbf{A}$ be an adversary making at most q enc_K and q_v dec_K queries with block length costs of at most σ_A, σ_P, and σ_C for associated, plaintext, and ciphertext data, respectively, then*

$$\mathsf{DAE}(\mathbf{A}) \leq \frac{N_{\mathsf{E}}^2}{2^{n+1}} + \frac{q_v}{2^n} + \frac{q^2}{2^n} + \frac{qq_v}{2^n} + \frac{(\sigma_P+\sigma_C)^2}{2^{n+1}} + \frac{4(\sigma_P+\sigma_C)}{2^n} + \frac{(4+\sigma_A+\sigma_P+\sigma_C)^2}{2^n} + \frac{4(q+q_v)^2}{2^n} + \mathsf{PRP}_{\mathsf{E}}(\mathbf{A}_{\mathsf{E}})\,. \tag{9}$$

In the following sections we go through the formal arguments of proving the above theorem. Sect. 4.11 finally summarizes all the results and computes the above bound.

4.3 Proof Notation

The size of a set S is indicated interchangeably by $|S|$ and $\#S$. Given sets J and X, the set of all mappings from J to X is denoted X^J. The set of all injective mappings is denoted $\partial\mathsf{X}^I$.

4.4 Switching to URFs.

Let $(\mathsf{enc}[F], \mathsf{dec}[F])$ represent SUNDAE's encryption and decryption algorithms with the block cipher calls E_K replaced by the function $F : \mathsf{B} \to \mathsf{B}$.

Lemma 1. *Let O_1, O_2 be any oracles, and let ρ be a URF over B. For any adversary $\mathbf{A}$ with block length costs of at most σ_A, σ_P and σ_C for associated data, plaintext, and ciphertext respectively, we have*

$$\underset{\mathbf{A}}{\Delta}(\mathsf{enc}[\mathsf{E}_K], \mathsf{dec}[\mathsf{E}_K]\,;\,O_1, O_2) \leq \underset{\mathbf{A}}{\Delta}(\mathsf{enc}[\rho], \mathsf{dec}[\rho]\,;\,O_1, O_2) + \frac{N_{\mathsf{E}}^2}{2^{n+1}} + \mathsf{PRP}_{\mathsf{E}}(\mathbf{A}_{\mathsf{E}})\,, \tag{10}$$

where $\mathbf{A}_{\mathsf{E}}$ is the standard model reduction described in Section 4.2, and N_{E} from Eq. (7) is an upper bound on the total number of block cipher calls $\mathbf{A}$ makes.

Proof. Let π denote a permutation chosen uniformly from the set of permutations over B. Applying the triangle inequality we get

$$\underset{\mathbf{A}}{\Delta}\left(\mathsf{enc}[\mathsf{E}_K], \mathsf{dec}[\mathsf{E}_K]\,;\, O_1, O_2\right) \leq \underset{\mathbf{A}}{\Delta}\left(\mathsf{enc}[\mathsf{E}_K], \mathsf{dec}[\mathsf{E}_K]\,;\, \mathsf{enc}[\pi], \mathsf{dec}[\pi]\right) + \underset{\mathbf{A}}{\Delta}\left(\mathsf{enc}[\pi], \mathsf{dec}[\pi]\,;\, \mathsf{enc}[\rho], \mathsf{dec}[\rho]\right) + \underset{\mathbf{A}}{\Delta}\left(\mathsf{enc}[\rho], \mathsf{dec}[\rho]\,;\, O_1, O_2\right) \quad (11)$$

The first term equals $\Delta_{\mathbf{A}_\mathsf{E}}\left(\mathsf{E}_K\,;\,\pi\right)$ and the second term $\Delta_{\mathbf{A}_\mathsf{E}}\left(\pi\,;\,\rho\right)$. The first term is exactly $\mathbf{A}_\mathsf{E}$'s PRP-advantage against E_K, and the second term is bounded above by the PRP-PRF switching lemma [BR06]. Knowing that $\mathbf{A}_\mathsf{E}$ makes at most N_E queries to its oracles, we have our desired bound. □

After applying the PRP-PRF switch, we have that $\mathsf{DAE}(\mathbf{A})$ is bounded by

$$\underset{\mathbf{A}}{\Delta}\left(\mathsf{enc}[\rho], \mathsf{dec}[\rho]\,;\, \$, \perp\right) + \frac{N_\mathsf{E}^2}{2^{n+1}} + \mathsf{PRP}_\mathsf{E}(\mathbf{A}_\mathsf{E})\,, \quad (12)$$

where ρ is a URF over B, $(\mathsf{enc}[\rho], \mathsf{dec}[\rho])$ is SUNDAE with block cipher calls replaced by ρ calls, and $\mathbf{A}$ may not query $\mathsf{dec}(A, C)$ after having queried $\mathsf{enc}(A, M)$ with output C. Hence we focus on $\mathbf{A}$'s advantage in distinguishing $(\mathsf{enc}[\rho], \mathsf{dec}[\rho])$ from \$.

4.5 Alternative Description of SUNDAE

For shorthand we denote $(\mathsf{enc}[\rho], \mathsf{dec}[\rho])$ by $(\mathsf{enc}, \mathsf{dec})$. The algorithm enc can be viewed as operating in three steps:

1. converting messages (A, M) to intermediate functions, followed by
2. applying the URF ρ to the intermediate functions to calculate an output stream, and finally
3. truncating the output stream and XORing it with M to create the ciphertext C.

Below we describe the above steps in detail, and Fig. 2 illustrates the steps in a diagram.

Step 1 of enc: From Messages to Intermediate Functions. The first step starts by splitting A and M into blocks, if non-empty, to get

$$A[1] \cdots A[\ell_A] \xleftarrow{n} A \text{ and } M[1] \cdots M[\ell_M] \xleftarrow{n} M\,. \quad (13)$$

Then each block is augmented with a bit to indicate whether it is a final block or not. We let split denote the operation of mapping (A, M) to the sequence of augmented blocks

$$\Big((0, A[1]), \ldots, (1, A[\ell_A]), (0, M[1]), \ldots, (1, M[\ell_M])\Big)\,. \quad (14)$$

The augmented blocks output by split are subsequently used as the first parameter in the function

$$f : \Big(\{0,1\} \times \{0,1\}^{\leq n}\Big) \times \mathsf{B} \to \mathsf{B}\,, \quad (15)$$

where f is defined as

$$f((\delta, X), Y) := \begin{cases} X \oplus Y & \text{if } \delta = 0 \\ 2 \times (\mathsf{pad}(X) \oplus Y) & \text{if } \delta = 1 \text{ and } |X| < n \\ 4 \times (X \oplus Y) & \text{otherwise} \end{cases}\,. \quad (16)$$

Recall that $f((\delta, X), Y)$ and $f_{\delta,X}(Y)$ are equivalent. We write the operation mapping an input (A, M) to a sequence of intermediate functions from B to B as $I(A, M)$. If $A \neq \varepsilon$ and $M \neq \varepsilon$, we have that

$$I(A,M) := \Big(110^{n-2}, f_{0,A[1]}, \cdots, f_{0,A[\ell-1]}, f_{1,A[\ell_A]}, \\ f_{0,M[1]}, \cdots, f_{0,M[\ell-1]}, f_{1,M[\ell_M]}\Big), \quad (17)$$

where values $X \in \{0,1\}^n$ are interpreted as constant functions mapping any element in B to X. Similarly, if M is non-empty,

$$I(\varepsilon, M) := \Big(010^{n-2}, f_{0,M[1]}, \cdots, f_{0,M[\ell-1]}, f_{1,M[\ell_M]}\Big), \quad (18)$$

if A is non-empty,

$$I(A, \varepsilon) := \Big(100^{n-2}, f_{0,A[1]}, \cdots, f_{0,A[\ell-1]}, f_{1,A[\ell_A]}\Big), \quad (19)$$

and finally $I(\varepsilon, \varepsilon) = (0^n)$. Let

$$IV_{A,M} := \begin{cases} 0^n & \text{if } A = \varepsilon, M = \varepsilon \\ 100^{n-2} & \text{if } A \neq \varepsilon, M = \varepsilon \\ 010^{n-2} & \text{if } A = \varepsilon, M \neq \varepsilon \\ 110^{n-2} & \text{otherwise .} \end{cases} \quad (20)$$

Step 2 of enc: Applying ρ. The algorithm's second step applies ρ to the sequence of intermediate functions specified by $I(A, M)$. Given $\vec{x} = (x_1, x_2, \ldots, x_\ell)$ where each x_i is a function from B to B, define the cascade of ρ with $\vec{x}$ to be the function $\widehat{\rho}$ from B to B defined by applying x_1 followed by ρ, followed by x_2, and so forth:

$$\widehat{\rho}(x_1, x_2, \ldots, x_\ell) = \rho \circ x_\ell \circ \rho \circ x_{\ell-1} \circ \cdots \circ \rho \circ x_3 \circ \rho \circ x_2 \circ \rho \circ x_1 . \quad (21)$$

Let $\mathsf{stream}_i(S_1) := S_1 S_2 \cdots S_{i+1}$, where $S_j = \rho(S_{j-1})$ for $1 < j \leq i+1$. We let $\mathsf{enc\text{-}stream}(A, M) := \mathsf{stream}_{\ell_M}\big(\widehat{\rho}(I(A, M))\big)$, representing the first and second steps of enc, where $\widehat{\rho}(I(A, M))$ is interpreted as an element of B.

Step 3 of enc: chopxor Define $\mathsf{chopxor}_Y(X)$ to be

$$\begin{cases} \lfloor X \rfloor_{|Y|+n} \oplus (0^n \| Y) & \text{if } X \neq \perp \\ \perp & \text{otherwise} \end{cases} . \quad (22)$$

Then $\mathsf{chopxor}_M(X)$ represents the final step of enc. Letting $\mathsf{enc\text{-}stream}$ denote the first two steps of enc, we have

$$\mathsf{enc}(A, M) = \mathsf{chopxor}_M\big(\mathsf{enc\text{-}stream}(A, M)\big) . \quad (23)$$

Using the above notation, we define

$$\mathsf{dec\text{-}stream}(A, TC) = \Big\lfloor \mathsf{enc\text{-}stream}\Big(A, \mathsf{chopxor}_C\big(\mathsf{stream}_{\ell_C}(T)\big)\Big)\Big\rfloor_n = T \ ? \\ \mathsf{stream}_{\ell_C}(T) : \perp , \quad (24)$$

where ℓ_C is the block length of C. Then dec can be described as

$$\mathsf{dec}(A, TC) = \mathsf{chopxor}_C\Big(\mathsf{dec\text{-}stream}(A, TC)\Big) . \quad (25)$$

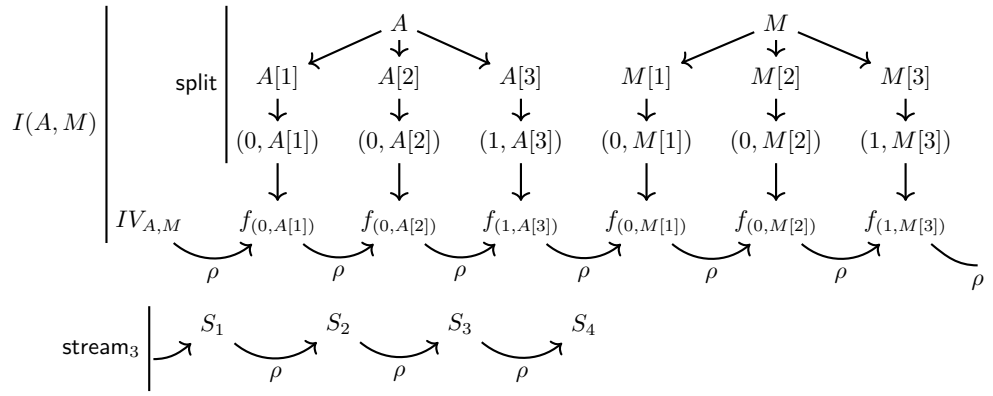


Figure 2: Schematic illustration of enc-stream.

4.6 Eliminating chopxor

Let

$$\$^s(A, M) := \$_{|A|,|M|,(\ell_M+1)*n}(A, M)\,, \tag{26}$$

that is, the enc-stream-equivalent of \$, then

$$\Delta\left(\$\,;\, \mathsf{chopxor}_M \circ \$^s\right) = 0\,. \tag{27}$$

Define $\mathbf{A}_{\mathsf{chopxor}}$ to be the adversary interacting with an oracle (O_1, O_2) — which is either (enc-stream, dec-stream) or $(\$^s, \perp)$ — that starts by running $\mathbf{A}$, and for each query $O_1(A, M)$ that $\mathbf{A}$ makes, $\mathbf{A}_{\mathsf{chopxor}}$ returns $\mathsf{chopxor}_M(O_1(A, M))$ to $\mathbf{A}$, and for each query $\mathsf{dec}(A, TC)$ made by $\mathbf{A}$, $\mathbf{A}_{\mathsf{chopxor}}$ returns $\mathsf{chopxor}_C(O_2(A, TC))$.

Then

$$\begin{aligned}
\underset{\mathbf{A}}{\Delta}\left(\mathsf{enc}, \mathsf{dec}\,;\, \$, \perp\right) &= \underset{\mathbf{A}}{\Delta}\left(\mathsf{enc}, \mathsf{dec}\,;\, \mathsf{chopxor}_M \circ \$^s, \perp\right) + \underset{\mathbf{A}}{\Delta}\left(\mathsf{chopxor}_M \circ \$^s, \perp\,;\, \$, \perp\right) && (28)\\
&= \underset{\mathbf{A}}{\Delta}\left(\mathsf{chopxor}_M \circ \mathsf{enc\text{-}stream}, \mathsf{chopxor}_C \circ \mathsf{dec\text{-}stream}\,;\, \mathsf{chopxor}_M \circ \$^s, \perp\right) && (29)\\
&\leq \underset{\mathbf{A}_{\mathsf{chopxor}}}{\Delta}\left(\mathsf{enc\text{-}stream}, \mathsf{dec\text{-}stream}\,;\, \$^s, \perp\right)\,, && (30)
\end{aligned}$$

where $\mathbf{A}_{\mathsf{chopxor}}$ never queries $\mathsf{dec\text{-}stream}(A, T \| \mathsf{chopxor}_M(S))$ after having made the queries $\mathsf{enc\text{-}stream}(A, M) = TS$. This means the bound on the distance between the oracles (enc-stream, dec-stream) and $(\$^s, \perp)$ with $\mathbf{A}_{\mathsf{chopxor}}$, is a bound on the distance between enc and \$ with $\mathbf{A}$. Hence we focus on bounding $\Delta_{\mathbf{A}_{\mathsf{chopxor}}}\left(\mathsf{enc\text{-}stream}, \mathsf{dec\text{-}stream}\,;\, \$^s, \perp\right)$.

4.7 Bounding Forgery Probability With dec-stream*

Note that dec-stream computes its output based on a query to enc-stream and a query to stream. If T is a tag previously output by enc-stream, then one can recreate $\mathsf{stream}(T)$ just by knowing what the previous enc-stream output is. If T is not a tag previously output by enc-stream, then with high probability $\mathsf{stream}(T)$ will just be a stream of uniform random output. The idea behind dec-stream* is to capture this behaviour: given only access to $\$^s$ and $\$^s$'s previous outputs, dec-stream* will try to mimic dec-stream's behaviour as closely as possible.

Each one of $\$^s$'s outputs $S_1, S_2, \ldots, S_q$ can be viewed as sequences of complete blocks $S_i[1]S_i[2]\cdots S_i[\ell_i] \xleftarrow{n} S_i$, hence given a value $X \in \mathsf{B}$, determining whether X has been

output by $\$^s$ is a question of finding i, j such that $X = S_i[j]$. If $j < \ell_i$, the block following X is defined to be $S_i[j+1]$, where i, j are the smallest indices such that $X = S_i[j]$.

The oracle $\mathsf{dec\text{-}stream}^*$ will have access to all of $\$^s$'s past outputs, and uses $\$^s$ and the function $\mathsf{stream}^*_\ell(T)$, defined as follows. On input (A, TC) with $|C| = \ell$, the algorithm stream^*_ℓ determines if T equals $\lfloor \$^s(A_i, M_i) \rfloor_n$ for the smallest such index i. If there is such an i, then stream^*_ℓ outputs the first $\ell * n$ bits of $\$^s(A_i, M_i)$. If ℓ is greater than $|\$^s(A_i, M_i)|$ then it pads with uniform random bits to reach an output length of $\ell * n$ bits. If there is no such i, then stream^* outputs $\ell * n$ uniform random bits. We add the requirement that $\mathsf{stream}^*_\ell(T)$ is a prefix of $\mathsf{stream}^*_{\ell'}(T)$ for all $\ell \leq \ell'$. Finally, $\mathsf{dec\text{-}stream}^*(A, TC)$ is defined as

$$\left\lfloor \$^s\Big(A, \mathsf{chopxor}_C\big(\mathsf{stream}^*_{\ell_C}(T)\big)\Big) \right\rfloor_n = T \quad ? \quad \mathsf{stream}^*_{\ell_C}(T) : \perp . \tag{31}$$

Then we have

$$\begin{aligned} \Delta_{\mathbf{A}_{\mathsf{chopxor}}}(\mathsf{enc\text{-}stream}, \mathsf{dec\text{-}stream}\,;\, \$^s, \perp) \leq \Delta_{\mathbf{A}_{\mathsf{chopxor}}}(\mathsf{enc\text{-}stream}, \mathsf{dec\text{-}stream}\,;\, \$^s, \mathsf{dec\text{-}stream}^*) \\ + \Delta_{\mathbf{A}_{\mathsf{chopxor}}}(\$^s, \mathsf{dec\text{-}stream}^*\,;\, \$^s, \perp) . \end{aligned} \tag{32}$$

Bounding

$$\Delta_{\mathbf{A}_{\mathsf{chopxor}}}(\$^s, \mathsf{dec\text{-}stream}^*\,;\, \$^s, \perp) , \tag{33}$$

is the same as bounding the probability that $\mathbf{A}_{\mathsf{chopxor}}$ forces $\mathsf{dec\text{-}stream}^*$ to output non-$\perp$ output when interacting with $(\$^s, \mathsf{dec\text{-}stream}^*)$. Consider adversary $\mathbf{A}^*$ given access to only $\$^s$, that runs $\mathbf{A}_{\mathsf{chopxor}}$, and forwards all of $\mathbf{A}_{\mathsf{chopxor}}$'s $\$^s$ queries to its own $\$^s$ oracle, and perfectly simulates $\mathsf{dec\text{-}stream}^*$ queries using $\$^s$.

The probability that two $\$^s$ queries collide in their first n bits of output (i.e., colliding tags) is at most $q^2/2^n$. Similarly, the probability that the first n bits of output of a new $\$^s$-query collides with the input to some past stream^*-query is at most $qq_v/2^n$. By excluding these bad events, each $\$^s$-output is uniquely identified by its first n bits, therefore if $\$^s(A, M) = TS$, and $C = \mathsf{chopxor}_M(S)$, then $\mathsf{stream}^*_{\ell_C}(T) = S$.

Say that $\mathbf{A}_{\mathsf{chopxor}}$ makes some query $\$^s(A, M) = TS$, then $\mathbf{A}_{\mathsf{chopxor}}$ cannot perform the $\mathsf{dec\text{-}stream}^*$ query $(A, T \| \mathsf{chopxor}_M(S))$, hence cannot implicitly call $\$^s(A, M)$ assuming $\mathsf{stream}^*_{\ell_C}(T) = S$. Therefore, after excluding the bad events above, if $\mathbf{A}_{\mathsf{chopxor}}$ succeeds in finding (A, TC) such that $\mathsf{dec\text{-}stream}^*(A, TC) \neq \perp$, then $\mathbf{A}^*$ has found a value $M := \mathsf{chopxor}_C(\mathsf{stream}^*_{\ell_C}(T))$ such that

1. (A, M) has never been queried to $\$^s$, and
2. $\lfloor \$^s(A, M) \rfloor_n = T$,

meaning $\mathbf{A}^*$ has either found a pre-image or a second pre-image for $\lfloor \$^s \rfloor_n$, which occurs with probability at most $q_v/2^n$, hence

$$\Delta_{\mathbf{A}_{\mathsf{chopxor}}}(\$^s, \mathsf{dec\text{-}stream}^*\,;\, \$^s, \perp) \leq \frac{q_v}{2^n} + \frac{q^2}{2^n} + \frac{qq_v}{2^n} . \tag{34}$$

4.8 Focusing on Transcripts

We are left with bounding

$$\Delta_{\mathbf{A}_{\mathsf{chopxor}}}(\mathsf{enc\text{-}stream}, \mathsf{dec\text{-}stream}\,;\, \$^s, \mathsf{dec\text{-}stream}^*) . \tag{35}$$

We apply Patarin's method, which we briefly review below.

4.8.1 Patarin's Method

When an adversary $\mathbf{A}$ interacts with an oracle $O : \mathsf{X} \to \mathsf{Y}$, it produces a sequence of inputs and outputs to the oracle

$$\vec{t} = ((x_1, y_1), \ldots, (x_q, y_q)) \in (\mathsf{X} \times \mathsf{Y})^* \,, \tag{36}$$

We let $O\langle\vec{t}\rangle$ denote the event that $O(x_i) = y_i$ for $i = 1, \ldots, q$, $\mathbf{A}\langle\vec{t}\rangle$ the event that $\mathbf{A}$ produces inputs $x_1, x_2, \ldots, x_q$ given oracle outputs $y_1, y_2, \ldots, y_q$, and $\mathbf{A}^O = \vec{t}$ the event that the interaction between $\mathbf{A}$ and O produces transcript $\vec{t}$. Note that in the events defined above, the order of the queries specified by the transcript $\vec{t}$ is important.

Adversarial advantage in distinguishing two oracles can be bounded by looking at the difference in transcript probabilitites. Initially formalized by Patarin [Pat91, Pat08], re-introduced by Chen and Steinberger [CS14], and to a certain extent independently discovered by Bernstein [Ber05], and Chang and Nandi [Nan06, CN08], the following lemma allows us to mostly focus on computing transcript probabilities to establish our results.

Lemma 2 (Patarin)**.** *Let $\mathbf{A}$ be an adversary attempting to distinguish oracle O_1 from oracle O_2, both with input domain X and output domain Y. Let $\mathsf{T} \subset (\mathsf{X} \times \mathsf{Y})^*$ denote the set of transcripts $\vec{t}$ such that $\mathbf{P}\left[\mathbf{A}\langle\vec{t}\rangle\right] > 0$, and say that T can be partitioned into a set of good transcripts T_{good} and a set of bad transcripts T_{bad}. If there exists ϵ such that for all $\vec{t} \in \mathsf{T}_{good}$,*

$$\mathbf{P}\left[O_1\langle\vec{t}\rangle\right] \geq (1-\epsilon) \cdot \mathbf{P}\left[O_2\langle\vec{t}\rangle\right] \,, \quad \textit{then} \quad \underset{\mathbf{A}}{\Delta}(O_1 \,;\, O_2) \leq \epsilon + \mathbf{P}\left[\mathbf{A}^{O_2} \in \mathsf{T}_{bad}\right] . \tag{37}$$

We apply Patarin's method by describing an event $\rho\text{-}\mathsf{coll}_{\vec{t}}$ such that for all $\vec{t}$ in some set T_{good},

$$\mathbf{P}\left[(\mathsf{enc\text{-}stream}, \mathsf{dec\text{-}stream})\langle\vec{t}\rangle \;\middle|\; \overline{\rho\text{-}\mathsf{coll}}_{\vec{t}}\right] = \mathbf{P}\left[(\$^s, \mathsf{dec\text{-}stream}^*)\langle\vec{t}\rangle\right] \,, \tag{38}$$

where $\overline{\rho\text{-}\mathsf{coll}}_{\vec{t}}$ is the complement of $\rho\text{-}\mathsf{coll}_{\vec{t}}$, so that

$$\mathbf{P}\left[(\mathsf{enc\text{-}stream}, \mathsf{dec\text{-}stream})\langle\vec{t}\rangle\right] \geq \mathbf{P}\left[(\$^s, \mathsf{dec\text{-}stream}^*)\langle\vec{t}\rangle\right] \cdot \mathbf{P}\left[\overline{\rho\text{-}\mathsf{coll}}_{\vec{t}}\right] . \tag{39}$$

If we can find ϵ and a set T_{good} such that $\mathbf{P}\left[\rho\text{-}\mathsf{coll}_{\vec{t}}\right] \leq \epsilon$ for all $\vec{t} \in \mathsf{T}_{\text{good}}$, then the above equation allows us to apply Lemma 2, which is Patarin's method. To arrive at this point, we introduce further terminology to describe the transcripts.

4.8.2 Transcript Description

A transcript $\vec{t}$ of $\mathbf{A}_{\mathsf{chopxor}}$'s interaction only contains $\mathsf{enc\text{-}stream}$ and $\mathsf{dec\text{-}stream}$ output, which hides the calls to stream made by $\mathsf{dec\text{-}stream}$; similarly, when $\mathbf{A}_{\mathsf{chopxor}}$ interacts with $(\$^s, \mathsf{dec\text{-}stream}^*)$, calls to stream^* are hidden. We augment $\mathbf{A}_{\mathsf{chopxor}}$'s transcripts to include the hidden output. As done in previous work [CS14, GPT15, MRV15], we release the hidden output to $\mathbf{A}_{\mathsf{chopxor}}$ after all queries have been made, but before $\mathbf{A}_{\mathsf{chopxor}}$ outputs its decision.

Each transcript consists of q O_1 queries (representing either $\mathsf{enc\text{-}stream}$ or $\$^s$), q_v O_2-queries (representing either $\mathsf{dec\text{-}stream}$ or $\mathsf{dec\text{-}stream}^*$), and q_v H-queries (representing either hidden stream or stream^* output), denoted

$$O_1(A_i^+, M_i^+) = T_i^+ S_i^+ \qquad \text{for } i = 1, \ldots, q \tag{40}$$

$$O_2(A_i^-, T_i^- C_i^-) = Y_i^- \qquad \text{for } i = 1, \ldots, q_v \tag{41}$$

$$H_{\ell_i^-}(T_i^-) = S_i^- \qquad \text{for } i = 1, \ldots, q_v \,, \tag{42}$$

where Y_i^- is either $\perp$ or a message, and ℓ_i^- is the block length of C_i^-. Note that in both worlds, $O_2(A, TC)$ can be written as

$$\Big\lfloor O_1\Big(A, \mathsf{chopxor}_C\big(H_{\ell_C}(T)\big)\Big)\Big\rfloor_n = T \;?\; H_{\ell_C}(T) : \perp\,. \tag{43}$$

Furthermore, the transcript's probability is non-zero only if $Y_i^- = \mathsf{chopxor}_{C_i^-}(S_i^-)$ for all successful forgeries, and $\perp$ otherwise. We can replace all O_2 queries by O_1 and H queries, and maintain the same transcript probability:

$$O_1(A_i^+, M_i^+) = T_i^+ S_i^+ \qquad \text{for } i = 1, \ldots, q \tag{44}$$

$$\lfloor O_1(A_i^-, M_i^-)\rfloor_n = T_i^- \qquad \text{for } i = 1, \ldots, q_v \tag{45}$$

$$H_{\ell_i^-}(T_i^-) = S_i^- \qquad \text{for } i = 1, \ldots, q_v\,, \tag{46}$$

where $M_i^- = \mathsf{chopxor}_{C_i^-}(S_i^-)$.

4.8.3 Viewing Transcripts as Graphs

Our goal is to have $\rho\text{-}\mathsf{coll}_{\vec{t}}$ describe the event that two ρ-inputs collide in a transcript $\vec{t}$. To do so, we extract the graph of queries made to ρ, illustrated for a single query in Fig. 2. App. B works through an example of the conversion from transcript to graph we describe in this section.

Definition 3. Given a set of sequences,

$$X_i = (X_i[1], X_i[2], \ldots, X_i[\ell_i]) \quad \text{for } i = 1, \ldots, q\,, \tag{47}$$

we define the *graph induced by the* X_i as follows. The nodes are all non-empty subsequences of the X_i, each labelled by their last element. Two nodes, v_1 and v_2, are connected by a directed edge $v_1 \to v_2$ if v_1 is a prefix of v_2, differing by one element.

Given a transcript $\vec{t}$, define $G_{\vec{t}}$ as the graph induced by the following sequences:

1. $(IV_{A_i^+, M_i^+}) \parallel \mathsf{split}(A_i^+, M_i^+)$ and $(IV_{A_i^-, M_i^-}) \parallel \mathsf{split}(A_i^-, M_i^-)$, and
2. the single-element sequences $\big((0, S_i^-[j])\big)$ and $\big((0, S_i^+[j])\big)$ for all i, j.

The graph $G_{\vec{t}}$ ideally captures the only information that the adversary should learn, namely that inputs to SUNDAE with the same prefixes will result in the same ρ-calls, but all outputs are unrelated to each other. Since SUNDAE internally maps each node of $G_{\vec{t}}$ to a ρ input, SUNDAE will maintain the graph's structure if no two ρ inputs collide.

SUNDAE maps $G_{\vec{t}}$ to ρ input via its intermediate functions, denoted f in Sect. 4.5. We introduce the graph $G_{\vec{t}}^I$ to describe the adversary's view after application of f: $G_{\vec{t}}^I$ is a graph induced by the following sequences:

1. $I(A_i^+, M_i^+)$ and $I(A_i^-, M_i^-)$, and
2. the single-element sequences $\big(S_i^-[j]\big)$ and $\big(S_i^+[j]\big)$, where each element is treated as a constant function mapping over B.

Note that $G_{\vec{t}}$'s nodes are sequences of blocks B, while $G_{\vec{t}}^I$'s nodes are sequences of functions defined on B.

There is a natural function induced by I mapping $G_{\vec{t}}$ to $G_{\vec{t}}^I$ as follows. Let v be a node in $G_{\vec{t}}$. If v is a sequence of length one of the form $((0, X))$, then map it to the node $((X))$ in $G_{\vec{t}}^I$. Otherwise find any $(A_i^\pm, M_i^\pm)$ such that v is a subsequence of $\mathsf{split}(A_i^\pm, M_i^\pm)$, then map v to the corresponding subsequence of $I(A_i^\pm, M_i^\pm)$. This mapping is well-defined

since if v is a subsequence of $(IV_{A_i^\pm, M_i^\pm}) \parallel \mathsf{split}(A_i^\pm, M_i^\pm)$ and $(IV_{A_j^\pm, M_j^\pm}) \parallel \mathsf{split}(A_j^\pm, M_j^\pm)$, then the corresponding subsequences in $I(A_i^\pm, M_i^\pm)$ and $I(A_j^\pm, M_j^\pm)$ will equal each other as well. Furthermore, edges between nodes are preserved since the mapping preserves subsequences.

Applying ρ to $G_{\vec{t}}^I$ means applying the cascade $\widehat{\rho}$ to all paths from root nodes to leaves, where each path is interpreted as a vector with root node as the first component, and leaf node as the last component. Given an arbitrary node v, its corresponding ρ-input is defined as v applied to the cascade of ρ with the sequence of nodes on the path connecting v to a root node. For example, say there is a path (v_0, v_1, v_2, v) connecting v to the root node v_0, then the ρ-input corresponding to v is defined as $v \circ \rho \circ v_2 \circ \rho \circ v_1 \circ \rho \circ v_0$, which we can interpret as an element of B since $v_0 \in \mathsf{B}$. The ρ-inputs corresponding to nodes in $G_{\vec{t}}^I$ are denoted by the label $\chi : v \mapsto \chi_v$, a random variable over B^V dependent on ρ.

4.8.4 Comparing Transcript Probabilities

Given a transcript $\vec{t}$, there are four types of collisions that could occur:

1. $\vec{t}$ contains an output block $S_i^\pm[j]$ with $j < \ell_i^\pm$ such that $S_i^\pm[j] \in \{0^n, 100^{n-2}, 010^{n-2}, 110^{n-2}\}$, allowing the adversary to determine ρ's output on any of those inputs,
2. $\vec{t}$ contains colliding output blocks, meaning i, i', j, j' such that either $i \neq i'$ or $j \neq j'$ and $S_i^\pm[j] = S_{i'}^\pm[j']$,
3. when mapping nodes from $G_{\vec{t}}$ to nodes in $G_{\vec{t}}^I$ two ρ-inputs inevitably collide through poor design of f, which we call a *structural* collision, and
4. when applying ρ to $G_{\vec{t}}^I$ two ρ-inputs collide, in which case we call the collision *accidental*.

Define the set T_{bad} so that it includes all transcripts satisfying conditions 1 and 2. We naturally have that T_{good} is the complement of T_{bad}. Proposition 2 below analyzes the probability of a bad transcript occuring when interacting with $(\$^s, \mathsf{dec\text{-}stream}^*)$. The last two events cannot be described purely in terms of $\vec{t}$ and lead to the following definition.

Definition 4 (ρ-$\mathsf{coll}_{\vec{t}}$)**.** Event ρ-$\mathsf{coll}_{\vec{t}}$ occurs if there are two different nodes v and w of $G_{\vec{t}}$ that map to two different nodes v' and w' in $G_{\vec{t}}^I$, respectively, under I's induced mapping, such that $\chi_{v'} = \chi_{w'}$.

As long as $\vec{t}$ is not in T_{bad}, each single-element sequence $((0, S_i^\pm[j]))$ gets placed as a distinct root node in $G_{\vec{t}}$ without children. Then, under the I-induced mapping, those single-element sequences get mapped to elements $((S_i^\pm[j]))$, which again become distinct root nodes in $G_{\vec{t}}^I$ without children. Finally, if ρ-$\mathsf{coll}_{\vec{t}}$ does not hold, then each ρ-output seen by the adversary is the result of ρ queried with an input which does collide with any other ρ-input, thereby establishing the following proposition, and hence also the statement given in (39).

Proposition 1.

$$\mathbf{P}\left[(\mathsf{enc\text{-}stream}, \mathsf{dec\text{-}stream})\langle\vec{t}\rangle \;\middle|\; \overline{\rho\text{-}\mathsf{coll}}_{\vec{t}}\right] = \mathbf{P}\left[(\$^s, \mathsf{dec\text{-}stream}^*)\langle\vec{t}\rangle\right]. \tag{48}$$

Proposition 2.

$$\mathbf{P}\left[\mathbf{A}_{\mathsf{chopxor}}^{(\$^s, \mathsf{dec\text{-}stream}^*)} \in \mathsf{T}_{bad}\right] \leq \frac{(\sigma_P + \sigma_C)^2}{2^{n+1}} + \frac{4(\sigma_P + \sigma_C)}{2^n}. \tag{49}$$

Proof. Since $\$^s$'s outputs are all uniform and independently distributed, the chance that one of its output blocks is in $\{0^n, 100^{n-2}, 010^{n-2}, 110^{n-2}\}$ is $4\sigma_P/2^n$, and the probability that two of its output blocks collide is $\sigma_P^2/2^{n+1}$. Similarly, $\mathsf{dec\text{-}stream}^*$ either repeats $\$^s$ output, or generates independent, uniform random output, in which case a bad transcript occurs with probability at most $4\sigma_C/2^n + \sigma_C^2/2^{n+1}$. □

4.9 Bounding ρ-coll$_{\vec{t}}$ Probability

As explained above, ρ-coll$_{\vec{t}}$ could occur either due to a structural or an accidental collision. A structural collision occurs when the I-induced mapping from $G_{\vec{t}}$ to $G^I_{\vec{t}}$ maps two different subsequences generated by split to the same subsequence generated by I. If two different message sequences get mapped to the same intermediate function sequence, then a ρ-input collision is guaranteed to occur if the ρ-input calculations start from the same constant. However, as long as the mappings $(\delta, X) \mapsto f_{\delta,X}$ are injective, meaning if $(\delta, X) \neq (\delta', X')$ then $f_{\delta,X} \neq f_{\delta',X'}$ as functions, the I-induced mapping will be injective as well.

Note that $(X, \delta) \mapsto f_{(X,\delta)}$ is in fact injective, since if $X \neq X'$, one can find Y such that

$$X \oplus Y \neq X' \oplus Y\,, \tag{50}$$
$$X \oplus Y \neq 2 \times (\mathsf{pad}(X') \oplus Y)\,, \text{ if } |X'| < n \text{ and} \tag{51}$$
$$X \oplus Y \neq 4 \times (X' \oplus Y) \text{if } |X'| = n\,. \tag{52}$$

Structural collisions only occur when the intermediate functions are not injective. Since we have chosen injective functions in our design such collisions never occur in transcripts produced by SUNDAE.

Nevertheless, there could still be accidental collisions among the ρ-inputs when ρ is applied to $G^I_{\vec{t}}$. In Sec. 4.10 we derive a general bound which can be used to analyze this case, which we subsequently apply.

For example, consider the transcript consisting of the following elements:

$$O_1(a_0||a_3, p_0||p_3) = s_0, s_1, s_2 \tag{53}$$
$$O_1(a_0||a_3, p_0||p_4) = s_3, s_4, s_5 \tag{54}$$
$$O_1(a_1, p_1||p_5) = s_6, s_7, s_8 \tag{55}$$
$$O_1(a_2||a_4, p_2) = s_9, s_{10} \tag{56}$$

For convenience we assume that all the a_i's and p_i's are mutually distinct and full blocks so that they are queried with the same initial IV.

In Figure 3, we construct the graphs $G_{\vec{t}}$ and $G^I_{\vec{t}}$ for this transcript, and illustrate all events of interest that can occur. The transcript is in T_{bad} if one of the 2 bad events occur in the induced graph $G^I_{\vec{t}}$:

1. either one of the s_i's is of the form $**0^{n-2}$, or
2. if for stream outputs of different O_1 queries we have $s_i = s_j$.

A structural collision occurs when we have some $f_{\delta,x} = f_{\delta',y}$ for $(\delta, x) \neq (\delta', y)$. In that case the structures of $G_{\vec{t}}$ and $G^I_{\vec{t}}$ are no longer isomorphic. For example in Figure 3, if $f_{0,a_0} = f_{1,a_1}$, then the nodes corresponding to this function collapse to one single node which is then connected to f_{0,p_1} by a dotted edge as shown in the figure. However the functions chosen in SUNDAE ensures that such collisions never occur.

The 4th type of collisions denoted by the event ρ-coll $_{\vec{t}}$ occurs when the labels of two different nodes χ_i and χ_j are accidentally equal due to the randomness induced by the URF ρ. In the following sub-section, we concentrate on finding the probability that this event occurs in a given directed graph $G^I_{\vec{t}}$ when ρ is chosen at random from all functions from $\mathsf{B} \to \mathsf{B}$.

4.10 Bounding ρ-Input Collisions

Let G be a graph which is a directed forest, meaning G consists of disjoint directed trees. Let V be the set of nodes of G and $R \subset V$ the set of root nodes. Say that each node in G is labelled by functions from X to X. We denote the function at a node $v \in V$ as $\hat{v}$.

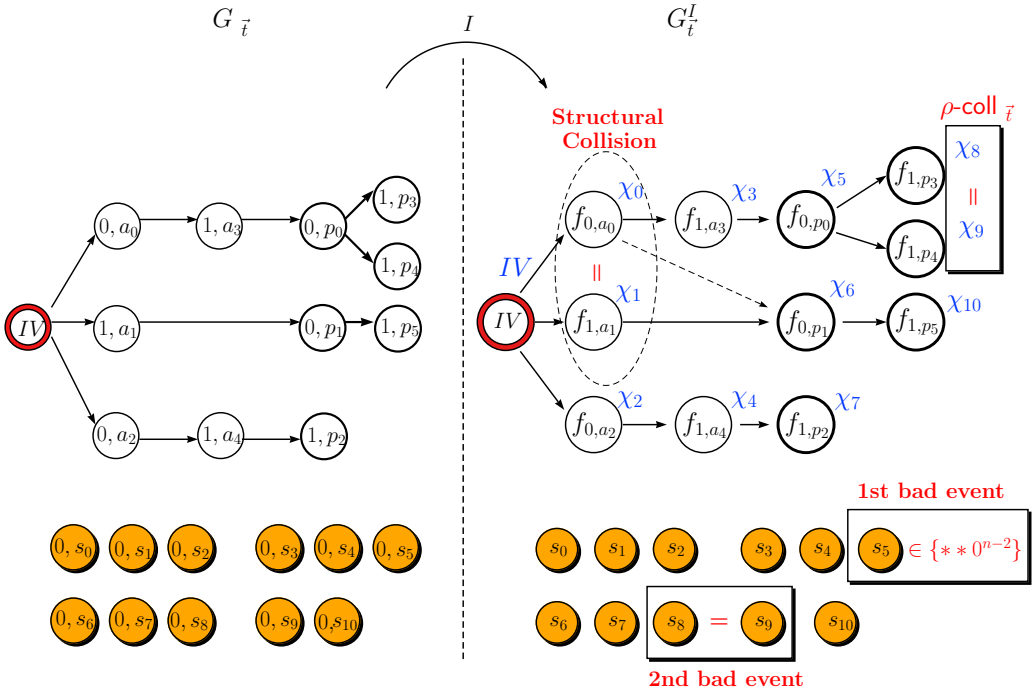


Figure 3: Illustrating the collisions that can happen.

Hence application of function at v to $x \in \mathsf{X}$ is written $\hat{v}(x)$. Furthermore, given a subset $S \subset V$, we write $\hat{S}$ to denote the set of functions underlying the nodes of S. In particular, $|\hat{S}| \leq |S|$, with equality if and only if each node in S represents a different function.

A sibling set S of G is the set of children of some other node in G. The only nodes in G which are not part of a sibling set are the root nodes. In particular, R along with all of G's sibling sets forms a partition of V.

As before, $\rho : \mathsf{X} \to \mathsf{X}$ can be applied to G by applying the cascade to each path in G starting from a root node. We let $\chi \in \mathsf{X}^V$ denote the ρ-inputs associated to the nodes V.

Consider the event that no two ρ-inputs collide, or equivalently, that χ is in $\partial\mathsf{X}^V$. Our goal is to characterize the probability of this event in terms of G as follows.

Proposition 3. *Let R denote G's root nodes, and say that $|\hat{R}| = |R|$. Let $\mathsf{S}_1, \mathsf{S}_2, \ldots, \mathsf{S}_\tau$ denote an enumeration of G's sibling sets, and let $N_i = |R| + \sum_{j=1}^{i-1} |\mathsf{S}_j|$. For $i > 0$ define*

$$\Gamma_i := \min_{\substack{\mathsf{Y} \subset \mathsf{X} \\ |\mathsf{Y}| = |X| - N_i}} \# \left\{ x \in \partial \mathsf{Y}^{\mathsf{S}_i} \;\middle|\; \bigcap_{w \in \mathsf{S}_i} \hat{w}^{-1}(x_w) \neq \emptyset \right\}, \tag{57}$$

then

$$\mathbf{P}\left[\chi \in \partial\mathsf{X}^V\right] \geq \frac{1}{|\mathsf{X}|^\tau} \prod_{i=1}^{\tau} \Gamma_i \,. \tag{58}$$

Proof. We know that

$$\mathbf{P}\left[\chi \in \partial\mathsf{X}^V\right] = \sum_{x \in \partial\mathsf{X}^V} \mathbf{P}\left[\chi = x\right], \tag{59}$$

therefore lower bounding this probability can be done by focusing on the labels in $\partial\mathsf{X}^V$ which occur with non-zero probability. We call such labels *valid*.

Since G's root nodes are fixed values, if two root nodes represent the same value, then no label in $\partial\mathsf{X}^V$ will be valid. Therefore we must use the fact that $|\hat{R}| = |R|$, since

otherwise it is impossible for no two ρ-inputs to collide. Furthermore, a label $x \in \partial \mathsf{X}^V$ is only valid if $x_v = \chi_v$ for all $v \in R$, hence we restrict our attention to such labels.

Since the only randomness present in χ is that provided by ρ, validity of a label x is determined via equations relating ρ to x, as imposed by G's edges. If node v is connected by an edge to node w, then w's label is calculated from v's label by applying ρ and then $\hat{w}$, or in other words, $\chi_w = \hat{w}(\rho(\chi_v))$, which is equivalent to $\rho(\chi_v) \in \hat{w}^{-1}(\chi_w)$. In particular, letting C_v denote the set of children of a node $v \in V$,

$$\mathbf{P}\left[\chi = x\right] = \mathbf{P}\left[\rho(x_v) \in \bigcap_{w \in C_v} \hat{w}^{-1}(x_w) \text{ for all } v \in V\right]. \tag{60}$$

Since $x_v \neq x_w$ for $v \neq w$, $\rho(x_v)$ is independent of $\rho(x_w)$ for all $w \neq v$, which means

$$\mathbf{P}\left[\chi = x\right] = \frac{1}{|\mathsf{X}|^\tau} \prod_{i=1}^{\tau} \left| \bigcap_{w \in \mathsf{S}_i} \hat{w}^{-1}(x_w) \right|, \tag{61}$$

where the children sets C_v have been replaced by the sibling sets $\mathsf{S}_1, \mathsf{S}_2, \ldots, \mathsf{S}_\tau$. Therefore the probability of a valid label can be lower bounded by $1/|\mathsf{X}|^\tau$ (which reaches equality if the functions $\hat{w}$ are bijective), and we get

$$\mathbf{P}\left[\chi \in \partial \mathsf{X}^G\right] \geq \frac{1}{|\mathsf{X}|^\tau} \cdot \#\left\{x \in \partial \mathsf{X}^V \;\middle|\; x \text{ is valid}\right\} \tag{62}$$

$$= \frac{1}{|\mathsf{X}|^\tau} \cdot \#\left\{x \in \partial \mathsf{X}^V \;\middle|\; \bigcap_{w \in \mathsf{S}_i} \hat{w}^{-1}(x_w) \neq \emptyset \text{ for all } i\right\}. \tag{63}$$

We lower bound the number of valid labels as follows. Consider the possible labels for the first sibling set S_1. We know that x_r must equal $\hat{r}$ for all root nodes $r \in R$. Since x_v with $v \in \mathsf{S}_1$ cannot equal x_r for $r \in R$, we know that $x_v \in \mathsf{X} \setminus \hat{R}$. Hence there are at most $\left|\partial(\mathsf{X} \setminus \hat{R})^{\mathsf{S}_1}\right|$ possible labellings of S_1, and at least Γ_1 valid ones. After fixing a labelling of S_1, the labels for S_2 must be taken from a set of size $|X| - |R| - |\mathsf{S}_1|$, and so Γ_2 lower bounds the number of possible valid labellings of S_2. Continuing like this for the other sibling sets, we have that Γ_i lower bounds the number of possible labellings for S_i, and the total number of valid labels for all of G is bounded below by the product of the Γ_i. □

Definition 5. For $w, w' \in V$ and $\mathsf{Y} \subset \mathsf{X}$, let $\mathsf{coll}(w, w') := \{\alpha \in \mathsf{X} \mid w(\alpha) = w'(\alpha)\}$.

Proposition 4. *If all functions in S_i are bijective for all i, then*

$$\mathbf{P}\left[\chi \notin \partial \mathsf{X}^V\right] \leq \frac{|V|^2}{|\mathsf{X}|} + \frac{1}{|\mathsf{X}|} \sum_{i=1}^{\tau} \left| \bigcup_{\substack{v,w \in \mathsf{S}_i \\ v \neq w}} \mathsf{coll}(v, w) \right|. \tag{64}$$

Proof. Let $w \in \mathsf{S}_i$ and consider some set $\mathsf{Y} \subset \mathsf{X}$ such that $|\mathsf{Y}| = |X| - N_i$. Since w is bijective, $\hat{w}^{-1}(x_w)$ is a singleton set, therefore for any $x \in \partial \mathsf{Y}^{\mathsf{S}_i}$ there is at most one element in the set $\bigcap_{w \in \mathsf{S}_i} \hat{w}^{-1}(x_w)$. Call the element α_x if it is present. Then the mapping from x for which α_x exists to α_x must be injective since $\hat{w}^{-1}$ is injective for all $w \in \mathsf{S}_i$. Furthermore, the mapping from $\alpha \in \mathsf{X}$ to $\partial \mathsf{X}^{\mathsf{S}_i}$ defined by $\alpha \mapsto (w \mapsto \hat{w}(\alpha))_{w \in \mathsf{S}_i}$ is injective as well since if $\alpha \neq \alpha'$, then $\hat{w}(\alpha) \neq \hat{w}(\alpha')$ for any $w \in \mathsf{S}_i$. Therefore

$$\Gamma_i = \min_{\mathsf{Y}} \#\left\{\alpha \in \mathsf{X} \;\middle|\; (w \mapsto \hat{w}(\alpha))_{w \in \mathsf{S}_i} \in \partial \mathsf{Y}^{\mathsf{S}_i}\right\}. \tag{65}$$

The set $\left\{\alpha \in \mathsf{X} \;\middle|\; (w \mapsto \hat{w}(\alpha))_{w \in \mathsf{S}_i} \in \partial \mathsf{Y}^{\mathsf{S}_i}\right\}$ can be rewritten as

$$\mathsf{Y}_i \setminus \{\alpha \in \mathsf{Y}_i \mid \exists v, w \in \mathsf{S}_i, v \neq w \text{ s.t. } \hat{v}(\alpha) = \hat{w}(\alpha)\}, \tag{66}$$

where $\mathsf{Y}_i = \bigcap_{w \in \mathsf{S}_i} \hat{w}^{-1}\mathsf{Y}$. In particular, $\Gamma_i \geq \min_{\mathsf{Y}} |\mathsf{Y}_i| - \# \bigcup_{\substack{v,w \in \mathsf{S}_i \\ v \neq w}} \mathsf{coll}(v,w)$.

Let $C_i := \# \bigcup_{\substack{v,w \in \mathsf{S}_i \\ v \neq w}} \mathsf{coll}(v,w)$. The size of Y_i is lower bounded in Lem. 3 in App. A, giving us $|\mathsf{Y}_i| \geq |X| - |\mathsf{S}_i|\, N_i$, hence

$$\Gamma_i \geq |X| - |\mathsf{S}_i|\, N_i - C_i = |X| \left(1 - \frac{|\mathsf{S}_i|\, N_i + C_i}{|X|}\right). \tag{67}$$

Applying the above to the collision probability of χ, we get

$$\mathbf{P}\left[\chi \notin \partial \mathsf{X}^V\right] = 1 - \mathbf{P}\left[\chi \in \partial \mathsf{X}^V\right] \leq 1 - \frac{1}{|\mathsf{X}|^\tau} \prod_{i=1}^{\tau} \Gamma_i \leq 1 - \prod_{i=1}^{\tau} \left(1 - \frac{|\mathsf{S}_i|\, N_i + C_i}{|\mathsf{X}|}\right) \tag{68}$$

$$\leq \sum_{i=1}^{\tau} \frac{|\mathsf{S}_i|\, N_i + C_i}{|\mathsf{X}|} \leq \sum_{i=1}^{\tau} \frac{|\mathsf{S}_i|\, N_i}{|\mathsf{X}|} + \sum_{i=1}^{\tau} \frac{C_i}{|\mathsf{X}|} \quad \leq \frac{|V|}{|\mathsf{X}|} \sum_{i=1}^{\tau} |\mathsf{S}_i| + \sum_{i=1}^{\tau} \frac{C_i}{|\mathsf{X}|} \tag{69}$$

$$\leq \frac{|V|^2}{|\mathsf{X}|} + \sum_{i=1}^{\tau} \frac{C_i}{|\mathsf{X}|} \tag{70}$$

□

Prop. 4 allows us to focus on analyzing collisions among the functions $(X, \delta) \mapsto f_{(X,\delta)}$. We have for any X and X', $\left|\mathsf{coll}_{\mathsf{X}}(f_{(X,0)}, f_{(X',1)})\right| \leq 1$, and if $|X| < n$ and $|X'| = n$, $\left|\mathsf{coll}_{\mathsf{X}}(f_{(X,1)}, f_{(X',1)})\right| \leq 1$, and if $X \neq X'$, $\mathsf{coll}_{\mathsf{X}}(f_{(X,0)}, f_{(X',0)}) = \emptyset$.

Then we upper bound $C_i = \# \bigcup_{\substack{v,w \in \mathsf{S}_i \\ v \neq w}} \mathsf{coll}(v,w)$. First, note that C_i is non-zero only if its associated sibling set has size greater than one. Sibling sets of size greater than one can only be created when a query is made, and for each query, either an existing sibling set becomes larger, or a new sibling set is created. This means there are at most $q + q_v$ sibling sets of size greater than one. The size of the sibling sets is also bounded above by $q + q_v$, therefore

$$\mathbf{P}\left[\rho\text{-}\mathsf{coll}_{\vec{t}}\right] \leq \frac{(4 + \sigma_A + \sigma_P + \sigma_C)^2}{2^n} + \frac{(q + q_v)^2}{2^n}. \tag{71}$$

4.11 Collecting the Results to Compute the Bound of Theorem 1

Sect. 4.4 applies the PRP-PRF switch to get that $\mathsf{DAE}(\mathbf{A})$ is bounded above by

$$\underset{\mathbf{A}}{\Delta}\left(\mathsf{enc}, \mathsf{dec}\,;\, \$, \bot\right) + \frac{N_{\mathsf{E}}^2}{2^{n+1}} + \mathsf{PRP}_{\mathsf{E}}(\mathbf{A}_{\mathsf{E}}), \tag{72}$$

allowing us to focus on $\Delta_{\mathbf{A}}\left(\mathsf{enc}, \mathsf{dec}\,;\, \$, \bot\right)$.

Sect. 4.6 proceeds to eliminate the $\mathsf{chopxor}$ function, concluding that

$$\underset{\mathbf{A}}{\Delta}\left(\mathsf{enc}, \mathsf{dec}\,;\, \$, \bot\right) \leq \underset{\mathbf{A}_{\mathsf{chopxor}}}{\Delta}\left(\mathsf{enc\text{-}stream}, \mathsf{dec\text{-}stream}\,;\, \$^s, \bot\right). \tag{73}$$

Then the intermediate construction $\mathsf{dec\text{-}stream}^*$ is introduced in Sect. 4.7, to establish that

$$\underset{\mathbf{A}_{\mathsf{chopxor}}}{\Delta}\left(\mathsf{enc\text{-}stream}, \mathsf{dec\text{-}stream}\,;\, \$^s, \bot\right)$$

$$\leq \underset{\mathbf{A}_{\mathsf{chopxor}}}{\Delta}\left(\mathsf{enc\text{-}stream}, \mathsf{dec\text{-}stream}\,;\, \$^s, \mathsf{dec\text{-}stream}^*\right) + \frac{q_v}{2^n} + \frac{q^2}{2^n} + \frac{q q_v}{2^n} \tag{74}$$

Using the bounds given in equations (72),(73), and (74), we have that the DAE security of SUNDAE is upper bounded by

$$\frac{N_{\mathsf{E}}^2}{2^{n+1}}+\mathsf{PRP}_{\mathsf{E}}(\mathbf{A}_{\mathsf{E}})+\frac{q_v}{2^n}+\frac{q^2}{2^n}+\frac{qq_v}{2^n}+\underset{\mathbf{A}_{\mathsf{chopxor}}}{\Delta}(\mathsf{enc\text{-}stream},\mathsf{dec\text{-}stream}\,;\,\$^s,\mathsf{dec\text{-}stream}^*)\ . \quad (75)$$

To bound the term $\Delta_{A_{chopxor}}(\mathsf{enc\text{-}stream},\mathsf{dec\text{-}stream};\$^s,\mathsf{dec\text{-}stream}^*)$ we make use of Patarin's method. To this end, we divided the set of transcripts seen by the adversary into good and bad transcripts. We have established in Sect. 4.8.4 that for all $\vec{t}\in\mathsf{T}_{\mathrm{good}}$,

$$\mathbf{P}\left[(\mathsf{enc\text{-}stream},\mathsf{dec\text{-}stream})\langle\vec{t}\rangle\right]\geq\mathbf{P}\left[(\$^s,\mathsf{dec\text{-}stream}^*)\langle\vec{t}\rangle\right]\cdot\mathbf{P}\left[\overline{\rho\text{-}\mathsf{coll}}_{\vec{t}}\right]\ . \quad (76)$$

By using (71), the above becomes

$$\mathbf{P}\left[(\mathsf{enc\text{-}stream},\mathsf{dec\text{-}stream})\langle\vec{t}\rangle\right]\geq\mathbf{P}\left[(\$^s,\mathsf{dec\text{-}stream}^*)\langle\vec{t}\rangle\right]\cdot\left(1-\frac{(4+\sigma_A+\sigma_P+\sigma_C)^2}{2^n}+\frac{4(q+q_v)^2}{2^n}\right)\ . \quad (77)$$

The probablity of a bad-transcript occuring when $\mathbf{A}_{\mathsf{chopxor}}$ is interacting with the intermediate oracles $\$^s$ and $\mathsf{dec\text{-}stream}^*$ is bounded above in Sect. 4.8.4

$$\frac{(\sigma_P+\sigma_C)^2}{2^{n+1}}+\frac{4(\sigma_P+\sigma_C)}{2^n}\,, \quad (78)$$

thus we can apply Lem. 2 in a straightforward manner to get

$$\underset{\mathbf{A}_{\mathsf{chopxor}}}{\Delta}(\mathsf{enc\text{-}stream},\mathsf{dec\text{-}stream}\,;\,\$^s,\mathsf{dec\text{-}stream}^*)\leq\frac{(\sigma_P+\sigma_C)^2}{2^{n+1}}+\frac{4(\sigma_P+\sigma_C)}{2^n}+\frac{(4+\sigma_A+\sigma_P+\sigma_C)^2}{2^n}+\frac{4(q+q_v)^2}{2^n}\,. \quad (79)$$

Thus adding the above bounds we get that the DAE security of SUNDAE is upper bounded by

$$\frac{N_{\mathsf{E}}^2}{2^{n+1}}+\mathsf{PRP}_{\mathsf{E}}(\mathbf{A}_{\mathsf{E}})+\frac{q_v}{2^n}+\frac{q^2}{2^n}+\frac{qq_v}{2^n}+\frac{(\sigma_P+\sigma_C)^2}{2^{n+1}}+\frac{4(\sigma_P+\sigma_C)}{2^n}+\frac{(4+\sigma_A+\sigma_P+\sigma_C)^2}{2^n}+\frac{4(q+q_v)^2}{2^n}\,. \quad (80)$$

5 Implementations

5.1 In Software: Embedded and Server-Side

SUNDAE is designed to have little overhead besides its block cipher calls. Besides an n-bit state, it only requires two XORs block and one or two finite field multiplications with a constant per message. Regarding performance, we expect serial software implementations of SUNDAE to run at half the speed of the underlying block cipher.

Setting. Our study considers embedded and high-performance parallel software implementation possibilities for SUNDAE with the following exemplary choices.

Block Cipher: We use AES [DR02] which is widely standardized and deployed in practice.

Platforms: As a case study for embedded devices, we use the Cortex-A57 core of a Samsung Exynos 7420 CPU (ARMv8 platform). For the server side, a Intel Core i7-6700 CPU (Skylake microarchitecture) was used. On both architectures, the cryptographic instruction support for AES is used, and key scheduling is precomputed. On each platform, SUNDAE is implemented serially with minimum overhead and in a parallel way for maximum performance.

Message Lengths: Performance data is provided for message lengths of $\ell = 2^b$ bytes, with $6 \leq b \leq 11$, covering most typical use cases, and in particular also illustrating SUNDAE's performance for relatively short inputs. To evaluate SUNDAE's efficiency when parallelization possibilities from multiple input streams arise, we also implemented it using the Comb scheduling strategy [BLT15] when instantiated with both fixed length messages and a message length mix according to a typical Internet packet size distribution [BLT15], in which around 40% of all packet lengths are short (below 100 bytes) and another 40% are moderately long (around 1500 bytes), hence emphasizing the importance of good performance for shorter messages.

Performance measurements. All measurements were taken on a single core. For the Intel platform, the CPU was a Core i7-6700 CPU at 3.4 GHz with Turbo Boost disabled. On ARM, a single Cortex-A57 core of the Samsung Exynos 7420 was used. The reported performance numbers were obtained as the median of 91 averaged timings of 200 measurements each [KR11]. The performance of AES, both in serial and parallel implementations, is provided as a baseline. Our implementation results are summarised in Table 1. All performance numbers are given in cycles per byte (cpb). The last column, denoted "mix", gives performance for the Internet message length distribution outlined previously.

Discussion. Table 1 confirms that serial software implementations of SUNDAE achieve roughly half the throughput of the underlying block cipher with almost no extra overhead on both Intel and ARM platforms: on Intel, SUNDAE is around 3% slower than two passes of CBC; on ARM, around 7%. SUNDAE's performance for short message lengths is only around 11% worse than for longer messages. Compared to the single-pass nonce-dependent COFB, SUNDAE has an overhead of 60% for short and 80% for long messages on Intel, and 35% for short and 80% for long messages on ARM.

Parallel implementations for processing multiple input streams are also possible, making full use of Intel and ARM's cryptographic instruction pipelines. Intel's AES-NI encryption instruction has a latency of 4 and an inverse throughput of 1 on the Skylake microarchitecture, whereas ARM's AES instructions come with a combined latency of 3 and an inverse throughput of 1 on the Cortex-A57. However unlike on Intel, they share the same pipeline as the logical and byte shuffling instructions (XOR,VEXT), which limits the performance gain for most block cipher modes, which alternate these with block cipher invocations.

As further reference points, we compare SUNDAE's performance to the nonce-dependent CLOC and JAMBU. As reported in [Iwa16], CLOC runs on Skylake at 2.82 cpb for long and 7.81 cpb for 64-byte messages. For short messages, SUNDAE performs better, whereas for longer messages, CLOC benefits from being a one pass, but two call, scheme, which allows limited use of the pipeline. Performance data for JAMBU on Skylake has been reported in [BLT16] as 5.5 cpb for long and 6.8 cpb for 64-byte messages, similar to SUNDAE's performance. However, recall that, unlike SUNDAE, CLOC, COFB and JAMBU all depend on nonce freshness for security.

Even in comparison with GCM-SIV, which is nonce misuse-resistant but targets high-end platforms, SUNDAE offers similar performance using the Comb technique: GCM-SIV runs at around 1.2 cpb for 2 KB messages [GL15], with SUNDAE performing at 1.3 cpb.

Table 1: Software performance of SUNDAE, COFB and CBC instantiated with AES on the ARMv8 and Intel Skylake platforms. Serial implementations are marked with (S), and parallel implementations processing multiple messages according to the Comb strategy with (P). All numbers are given in cycles per byte (cpb).

(a) Intel Skylake platform (server)

	message length (bytes)						
Algorithm	64	128	256	512	1024	2048	mix
CBC (S)	2.90	2.75	2.68	2.63	2.60	2.59	2.67
CBC (P)	0.64	0.64	0.63	0.63	0.63	0.63	0.64
COFB (S)	3.71	3.32	3.12	3.02	2.97	2.96	3.12
COFB (P)	1.03	0.95	0.90	0.87	0.86	0.85	0.90
SUNDAE (S)	6.00	5.71	5.57	5.46	5.40	5.37	5.52
SUNDAE (P)	1.36	1.31	1.29	1.27	1.26	1.26	1.28

(b) ARMv8 platform (embedded)

	message length (bytes)						
Algorithm	64	128	256	512	1024	2048	mix
CBC (S)	2.69	2.54	2.39	2.30	2.26	2.25	2.38
CBC (P)	1.42	1.14	1.02	0.95	0.92	0.90	1.00
COFB (S)	3.99	3.34	2.96	2.78	2.72	2.71	2.98
COFB (P)	2.98	1.89	1.49	1.32	1.25	1.22	1.52
SUNDAE (S)	5.42	5.14	5.02	4.92	4.86	4.84	4.97
SUNDAE (P)	3.16	2.95	2.85	2.80	2.78	2.76	2.84

5.2 ASIC Implementation

Lightweight implementations of CLOC [IMGM14], SILC [IMG+14] and AES-OTR [Min14], with AES-128 as the underlying block cipher has already been published in [BBM16]. As with all rate 1/2 modes like CLOC, SILC, and also some rate 1 modes like OTR [BBM16] there is a need for offline storage of message blocks for reading them twice. Note that this was also assumed in the lightweight implementation of the above modes in [BBM16]. In this work, the authors use the 8-bit serial implementation of AES given in [MPL+11]. The authors implemented the above modes for two typical use cases **(a)** aggressive and **(b)** conservative. The aggressive design implemented a version of the circuit that only catered to a limited set of sizes of the plaintext and associated data. For example, the aggressive circuit was only designed to process user inputs in which the associated data was empty and the length of the plaintext was an integral multiple of the block size of the underlying block cipher. The intermediate outputs produced by circuit were stored offline, and an external processor made them available at the input buses as required by the design. This relaxed many of the storage requirements in the circuit, and so the circuit occupied lower gate area. The conservative circuit had no such constraints and was designed to handle all types of user inputs within certain bounds (upto 8 blocks of associated data and 256 blocks of plaintext). All outputs of intermediate modules were stored in additional registers in the circuit. As a result of which its gate area was significantly larger.

In our implementation of SUNDAE, we do not distinguish between different use cases: our implementation is able to handle inputs of all sizes within the aforementioned bounds. Furthermore, the mode of operation has been designed in a manner that does not require temporary storage of any intermediate results, as a result no additional storage elements are required for this purpose. This essentially corresponds to the conservative use case of [BBM16]. We implement the mode using the block ciphers AES-128 and Present [BKL+07]. For AES, we use the Atomic-AES architectures developed in [BBR16a,BBR16b]. These are 8-bit serial architectures for AES meant for accommodating both encryption/decryption on the same platform. We use the Atomic AES v2.0 architecture which is smaller in area than the circuit in [MPL+11] by around 200 gate equivalents (GE)[1]. Furthermore, we try to do away with the requirement of an additional register to perform field doublings and quadruplings by changing the structure of the finite field. In stead of performing doubling over $GF(2^{128})$, we perform 8 doublings over $GF(2^{16})/ < x^{16} + x^5 + x^3 + x + 1 >$ in the following way. If $c_0, c_1, \ldots, c_{15}$ denote the individual bytes of the state (taken in a row-major fashion from the 4×4 state array), then we consider the string of i^{th} bits of each of these bytes $c_0[i], c_1[i], \ldots, c_{15}[i]$ as an element of of $GF(2^{16})$ in the polynomial basis described above, and perform doubling over this field for all the 8 values of $i \in [0, 7]$. Hence, the function f mapping this transition is simply

$$f(c_0, c_1, \ldots, c_{15}) = c_1, c_2, \ldots, c_{11} \oplus c_0, c_{12}, c_{13} \oplus c_0, c_{14}, c_{15} \oplus c_0, c_0$$

Note that f now becomes a function that can be easily implemented using a bytewise shift register as shown in Figure 4. The diagram shows a glimpse of the datapath of the design. The numbered boxes denote byte sized registers, and those colored grey denote scan registers. We refer to [BBR16b] for a detailed functional description of the circuit. As can be seen, in addition to the original AES circuit, only three 8-bit xor gates, an 8-bit two-input multiplexer, and an 8-bit two-input AND gate are required.

When the signal FMODE is low, the output of the AND gate is zero and all the xor gates are essentially bypassed (logically), and the circuit behaves as it should during the encryption cycles, with data flowing along the blue path between the registers. However when FMODE is high, the circuit computes f in the next clock cycle. Since quadrupling is $f \circ f$, it can be computed by setting FMODE to logic high for two consecutive cycles.

[1] 1 Gate equivalent (GE) is the area occupied by a 2 input NAND gate

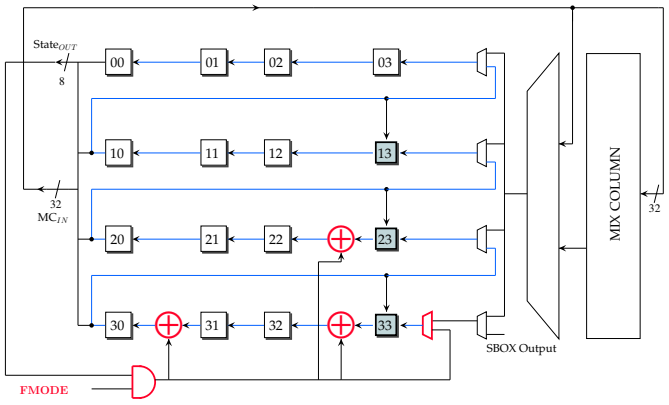


Figure 4: Datapath of SUNDAE (the figures in red denote the additional components required in the circuit)

Timing. The Atomic-AES architecture takes 246 clock cycles to encrypt one block of plaintext. Note that the first 16 cycles are used for loading the plaintext/key on to the registers. Again, the last 16 cycles are used to produce the ciphertext in a bytewise manner. Therefore if the mode of operation calls for 2 consecutive encryption operations (like in lines 8,17,23 of Algorithm 1), then in the last 16 cycles of the 1st encryption, the intermediate ciphertext can be xored with the associated data/plaintext and loaded on to the registers to start the next encryption cycle. As a result, a total of n consecutive encryptions can be done in $246 + (n-1) \cdot 230$ cycles. Let L^* be the number of blocks in the associated data and the initial $b_1||b_2||0^{n-2}$ block, and L be the number of blocks in the plaintext. We break up the analysis into subcases:

1. **Both** $|A| = |M| = 0$**:** In this case $L^* = 1$ and $L = 0$. Only one block is encrypted which takes 246 cycles.

2. $|A| = 0$ **but** $|M| \neq 0$**:** In this case $L^* = 1$ and $L \geq 1$. The initial $b_1||b_2||0^{n-2}$ block and the first $L-1$ plaintext blocks are processed in $T_2 = 246 + (L-1) \cdot 230$ cycles. Then $F_2 = 1$ or 2 cycles are used for the doubling/quadrupling. The last plaintext block and all the plaintext blocks in the second pass require $T_3 = 246 + L \cdot 230$ cycles. The total time taken to process one user input is $T_2 + T_3 + F_2$ cycles.

3. $|A| \neq 0$ **but** $|M| = 0$**:** In this case $L^* \geq 2$ and $L = 0$. The initial $L^* - 1$ blocks can be encrypted in $T_1 = 246 + (L^* - 2) \cdot 230$ cycles. This is interrupted by the field doubling/quadrupling which takes $F_1 = 1$ or 2 cycles according as the last associated data block is non-integral or not. Thereafter, last associated data block takes another $T_2 = 246$ cycles. The total time taken is $T_1 + F_1 + T_2$ cycles.

4. **Both** $|A| \neq 0$ **and** $|M| \neq 0$**:** In this case $L^* \geq 2$ and $L \geq 1$. The initial $L^* - 1$ blocks can be encrypted in $T_1 = 246 + (L^* - 2) \cdot 230$ cycles. This is interrupted by the field doubling/quadrupling which takes $F_1 = 1$ or 2 cycles according as the last associated data block is non-integral or not. Thereafter, last associated data block and the first $L-1$ plaintext blocks are processed in $T_2 = 246 + (L-1) \cdot 230$ cycles. Then $F_2 = 1$ or 2 cycles are used for the doubling/quadrupling. The last plaintext block and all the plaintext blocks in the second pass require $T_3 = 246 + L \cdot 230$ cycles. So the total time taken to process one user input is $T_1 + T_2 + T_3 + F_1 + F_2$ cycles.

Present. For the implementation with Present as the underlying Block cipher, we use the 4-bit serial implementation proposed in [RPLP08]. Since this is a nibble based implementation of the Present block cipher, we define the doubling and quadrupling operations in a nibble wise fashion. That is to say if $d_0, d_1, \ldots, d_{15}$ are the 16 nibbles of the 64-bit block, we perform 4 doublings over $GF(2^{16})/ < x^{16}+x^5+x^3+x+1 >$ for each of the 4 bit-strings $d_0[i], d_1[i], \ldots, d_{15}[i]$ for $i \in [0,3]$. Also one encryption using the nibble serial architecture in Present takes 567 cycles, out of which 20 cycles are needed to load the 80-bit key in a nibble-wise fashion. Hence, n consecutive encryptions take $567+(n-1)\cdot 547$ cycles, and the expressions for T_1, T_2, T_3 change accordingly. It is obvious that the construction with Present offers much less throughput than AES. However, Present is one of the most well analyzed lightweight block ciphers and has been adopted as a standard in ISO/IEC 29192-2. As seen in Table 2, the total circuit area for Present-SUNDAE is only around 1450 gates, which makes it ideal for lightweight platforms.

Results. In Table 2 we present the synthesis results for the designs. The following design flow was used: first the design was implemented in VHDL. Then, a functional verification was first done using Mentor Graphics Modelsim software. The designs were synthesized using the standard cell library of the 90nm logic process of STM (CORE90GPHVT v 2.1.a) with the Synopsys Design Compiler, with the compiler being specifically instructed to optimize the circuit for area. A timing simulation was done on the synthesized netlist. The switching activity of each gate of the circuit was collected while running post-synthesis simulation. The average power was obtained using *Synopsys Power Compiler*, using the back annotated switching activity.

As can be seen in the table, our implementation of AES-SUNDAE occupies around 2524 GE and is around 600 GE smaller than the aggressively designed CLOC and SILC circuits, synthesized with the same standard cell library. Since it is only fair to compare our design with the conservative CLOC/SILC/AES-OTR designs, we see that here too our implementation outperforms the CLOC, SILC and AES-OTR circuits by around 1800, 1700 and 4220 GE respectively. Also, Present-SUNDAE only occupies around 1452 GE which makes it ideal for deployment in lightweight platforms.

In Table 3, we present the area-wise breakup of the various components of the circuit. For AES-SUNDAE we see that around 85% of the area is occupied by the encryption logic (AES core) alone. This area includes the three additional circuit elements to compute the field doubling. Around 4% of the area is required for the length counters that encode and keep track of the number of blocks of associated data and plaintext currently processed. The remaining area is required for the control logic for routing signals to and out of the encryption core. For Present-SUNDAE, we have roughly the same area distribution, but the percentage contributions are different since the Present core is much smaller than AES.

Table 2: Implementation results for CLOC, SILC, AES-OTR, and SUNDAE. (Power reported at 10 MHz, A: Aggressive, C: Conservative

Block Cipher	Mode	Area (GE)	Power(μW)
AES-128	CLOC (A)	3110	131.1
	CLOC (C)	4310	156.6
	SILC (A)	3110	131.0
	SILC (C)	4220	155.6
	OTR (A)	4720	164.3
	OTR (C)	6770	205.4
	SUNDAE	**2524**	126.1
Present-80	SUNDAE	**1452**	50.9

Table 3: Componentwise breakdown of the circuit areas for SUNDAE.

Block Cipher	Core	Length Counters	Control Logic
AES-128	2145 GE 85%	100 GE 4%	279 GE 11%
Present-80	1082 GE 74.5%	100 GE 6.9%	272 GE 18.6%

5.2.1 Comparison with JAMBU and COFB

These figures also compare favorably with modes like JAMBU. The state size in JAMBU is one and a half times the block size of underlying block cipher. So when implemented with AES-128 as the underlying cipher, the mode requires a state of 192 bits. Only 128 bits can be accommodated in the data register of AES and so an additional 64 bit register is needed, which requires around 300 GE. So, if we use the Atomic-AES architecture to design the circuit for JAMBU we can estimate that the area required would be approximately 2100 (AES core) + 100 (Length counter) + 250 (control logic) + 300 (64 bit Register) $\approx$ 2750 GE, which is still around 250 GE more than the circuit for AES-SUNDAE.

COFB [CIMN17b, CIMN17a] is an AE mode designed by Chakraborti et al. at CHES 2017. It aims at reducing the hardware area over and above the underlying block cipher by using an $\frac{n}{2}$ size mask (where n is the block size of the underlying block cipher). Since most AE modes use an n bit mask this reduces the storage requirement by $\frac{n}{2}$ flip-flops over many standard AE schemes. However, this still needs an additional register of $\frac{n}{2}$ bits to store and update the mask, and so COFB like JAMBU has an effective internal state of $1.5n$ bits. Even so, the design is suited for lightweight implementation using the Atomic-AES architecture. We note the following:

- **Mask Logic:** To implement COFB, we need a mask register of size $\frac{n}{2} = 64$ bits. The register needs to be initialized by an intermediate 64-bit value Δ, and 3 possible updates are specified: (a) Multiplication by the primitive element α of $GF(2^{64})/ < x^{64}+x^4+x^3+x+1 >$, (b) Multiplication by $1+\alpha$ and (c) Multiplication by $(1+\alpha)^2$. Multiplication by α will need only 3 xor gates, whereas multiplication by $1+\alpha$ requires $64+3 = 67$ xor gates. Multiplication by $(1+\alpha)^2$ can be achieved by doing multiplication by $1+\alpha$ over 2 successive cycles. So at different points of time in the encryption cycle, the register would have to load one of 3 values: the constant Δ, the result of multiplication by α or $1+\alpha$. The most hardware effective way of implementing it is using a register with scan flip-flops along with an additional 64 bit multiplexer. So in total, we can estimate that the mask logic will require around 350 (scan register) + 128 (multiplexer) + 134 (xor gates) = 612 GE.
- **Linear Mixing:** COFB uses the linear mixing function G over the 128 bit AES state. This function is essentially same as the g_2 function used in CLOC. As a result, the AES state register can be tweaked like a columnwise shift register to achieve this functionality, exactly the same way as it was done in [BBM16, Fig. 1(a)]. The tweak requires a 32 bit multiplexer and an equal number of xor gates. Thus the modified AES circuit would require 2060 + 64 (multiplexer)+ 64 (xor gates) =2188 GE.

Therefore we can estimate that COFB will require approximately 2188 (AES core) + 100 (Length counter) + 250 (control logic) + 612 (Mask Register) $\approx$ 3150 GE.

5.2.2 Throughput

We can also estimate the throughput of the various modes when using the Atomic AES architecture, using the fact that n consecutive blocks are encrypted in $246 + 230(n-1)$ cycles. We tabulate the number of cycles-per-byte (CpB) required to operate the modes as a function of message length in Table 4. Among all rate 1/2 modes, AES-SUNDAE performs best when dealing with short messages less than 16 plaintext blocks, whereas all rate 1/2 modes asymptotically reach a constant CpB value for longer messages. Among the rate 1 modes, the performance of SUNDAE is comparable with OTR for short messages of 1 to 4 blocks, whereas COFB performs best overall.

Table 4: Throughput in (CpB) as a function of message length, using the Atomic AES architecture

		Message length (bytes)										
Mode	Rate	16	32	64	128	256	512	1024	2048	4096	8192	16384
CLOC (C)	1/2	63.8	46.3	37.5	33.1	30.9	29.8	29.3	29.0	28.9	28.8	28.8
SILC (C)	1/2	95.8	62.3	45.5	37.1	32.9	30.8	29.8	29.3	29.0	28.9	28.8
AES-OTR (C)	1	77.1	45.8	30.3	22.6	18.7	16.8	15.8	15.4	15.1	15.0	14.9
AES-JAMBU	1/2	72.8	50.8	39.8	34.3	31.5	30.1	29.4	29.1	28.9	28.8	28.8
AES-COFB	1	47.3	31.3	23.4	19.4	17.4	16.4	15.9	15.7	15.6	15.5	15.5
AES-SUNDAE	1/2	60.8	44.8	36.8	32.8	30.8	29.8	29.3	29.0	28.9	28.8	28.8

References

[ABB+14a] Elena Andreeva, Begül Bilgin, Andrey Bogdanov, Atul Luykx, Florian Mendel, Bart Mennink, Nicky Mouha, Qingju Wang, and Kan Yasuda. PRIMATEs v1, 2014. Submission to CAESAR competition.

[ABB+14b] Elena Andreeva, Begül Bilgin, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. APE: Authenticated permutation-based encryption for lightweight cryptography. In Cid and Rechberger [CR15].

[ABL+13] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Elmar Tischhauser, and Kan Yasuda. Parallelizable and authenticated online ciphers. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I*, volume 8269 of *Lecture Notes in Computer Science*, pages 424–443. Springer, 2013.

[ABL+14] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. How to securely release unverified plaintext in authenticated encryption. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 105–125. Springer, 2014.

[ADL17] Tomer Ashur, Orr Dunkelman, and Atul Luykx. Boosting authenticated encryption robustness with minimal modifications. 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 3–33. Springer, 2017.

[ÅHJM11] Martin Ågren, Martin Hell, Thomas Johansson, and Willi Meier. Grain-128a: a new version of grain-128 with optional authentication. *IJWMC*, 5(1):48–59, 2011.

[AJN14] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. NORX v1, 2014. Submission to CAESAR competition.

[BBI+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A block cipher for low energy. In Iwata and Cheon [IC15], pages 411–436.

[BBM16] Subhadeep Banik, Andrey Bogdanov, and Kazuhiko Minematsu. Low-area hardware implementations of CLOC, SILC and AES-OTR. In William H. Robinson, Swarup Bhunia, and Ryan Kastner, editors, *HOST*, pages 71–74. IEEE Computer Society, 2016.

[BBR16a] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. Atomic-AES: A Compact Implementation of the AES Encryption/Decryption Core. In Orr Dunkelman and Somitra Kumar Sanadhya, editors, *INDOCRYPT*, volume 10095 of *Lecture Notes in Computer Science*, pages 173–190, 2016.

[BBR16b] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. Atomic-AES v 2.0. *IACR Cryptology ePrint Archive*, 2016:1005, 2016.

[BDJR97] Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 394–403. IEEE Computer Society, 1997.

[BDP+14a] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Ketje v1, 2014. Submission to CAESAR competition.

[BDP+14b] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Keyak v1, 2014. Submission to CAESAR competition.

[BDPV11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography*, volume 7118 of *Lecture Notes in Computer Science*, pages 320–337. Springer, 2011.

[Ber05] D. J. Bernstein. A short proof of the unpredictability of cipher block chaining, 2005. Document ID: 24120a1f8b92722b5e15fbb6a86521a0.

[BJK+16] Christof Beierle, J'er'emy Jean, Stefan K"olbl, 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 Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 123–153. Springer, 2016.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[BKL+13] Andrey Bogdanov, Miroslav Knezevic, Gregor Leander, Deniz Toz, Kerem Varici, and Ingrid Verbauwhede. SPONGENT: the design space of lightweight cryptographic hashing. *IEEE Trans. Computers*, 62(10):2041–2053, 2013.

[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.

[BLT16] Andrey Bogdanov, Martin M. Lauridsen, and Elmar Tischhauser. Comb to pipeline: Fast software encryption revisited. *IACR Cryptology ePrint Archive*, 2016:47, 2016.

[BMR+13] Andrey Bogdanov, Florian Mendel, Francesco Regazzoni, Vincent Rijmen, and Elmar Tischhauser. ALE: aes-based lightweight authenticated encryption. In Shiho Moriai, editor, *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, volume 8424 of *Lecture Notes in Computer Science*, pages 447–466. Springer, 2013.

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Vaudenay [Vau06], pages 409–426.

[BRW04] Mihir Bellare, Phillip Rogaway, and David Wagner. The EAX mode of operation. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*, pages 389–407. Springer, 2004.

[BZD+16] Hanno Böck, Aaron Zauner, Sean Devlin, Juraj Somorovsky, and Philipp Jovanovic. Nonce-disrespecting adversaries: Practical forgery attacks on GCM in TLS. In *10th USENIX Workshop on Offensive Technologies, WOOT 16, Austin, TX, August 8-9, 2016*. USENIX Association, 2016.

[CAE16] CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. http://competitions.cr.yp.to/caesar.html, 2016. Accessed: 2017-05-16.

[CDK09] Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - A family of small and efficient hardware-oriented block ciphers. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, volume 5747 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2009.

[CIMN17a] Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, and Mridul Nandi. Blockcipher-based authenticated encryption: How small can we go? In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei,*

Taiwan, September 25-28, 2017, Proceedings, Lecture Notes in Computer Science, page 21. Springer, 2017. To appear.

[CIMN17b] Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, and Mridul Nandi. Blockcipher-based authenticated encryption: How small can we go? Cryptology ePrint Archive, Report 2017/649, 2017. http://eprint.iacr.org/2017/649.

[CN08] Donghoon Chang and Mridul Nandi. A short proof of the PRP/PRF switching lemma. *IACR Cryptology ePrint Archive*, 2008:78, 2008.

[CP08] Christophe De Cannière and Bart Preneel. Trivium. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 244–266. Springer, 2008.

[CR15] Carlos Cid and Christian Rechberger, editors. *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, volume 8540 of *Lecture Notes in Computer Science*. Springer, 2015.

[CS14] Shan Chen and John P. Steinberger. Tight security bounds for key-alternating ciphers. In Nguyen and Oswald [NO14], pages 327–350.

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

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.

[Dun09] Orr Dunkelman, editor. *Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers*, volume 5665 of *Lecture Notes in Computer Science*. Springer, 2009.

[FFL12] Ewan Fleischmann, Christian Forler, and Stefan Lucks. Mcoe: A family of almost foolproof on-line authenticated encryption schemes. In Anne Canteaut, editor, *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*, pages 196–215. Springer, 2012.

[GL15] Shay Gueron and Yehuda Lindell. GCM-SIV: full nonce misuse-resistant authenticated encryption at under one cycle per byte. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 109–119. ACM, 2015.

[GPP11] Jian Guo, Thomas Peyrin, and Axel Poschmann. The PHOTON family of lightweight hash functions. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 222–239. Springer, 2011.

[GPT15] Peter Gazi, Krzysztof Pietrzak, and Stefano Tessaro. The exact PRF security of truncation: Tight bounds for keyed sponges and truncated CBC. In Gennaro and Robshaw [GR15], pages 368–387.

[GR15] Rosario Gennaro and Matthew Robshaw, editors. *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*. Springer, 2015.

[HRRV15] Viet Tung Hoang, Reza Reyhanitabar, Phillip Rogaway, and Damian Viz'ar. Online authenticated-encryption and its nonce-reuse misuse-resistance. In Gennaro and Robshaw [GR15], pages 493–517.

[IC15] Tetsu Iwata and Jung Hee Cheon, editors. *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*. Springer, 2015.

[IMG+14] Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, Sumio Morioka, and Eita Kobayashi. SILC: SImple Lightweight CFB. 2014. https://competitions.cr.yp.to/round1/silcv1.pdf.

[IMGM14] Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, and Sumio Morioka. CLOC: authenticated encryption for short input. In Cid and Rechberger [CR15], pages 149–167.

[Iwa16] Tetsu Iwata. Updates on CLOC and SILC Version 3. In *DIAC 2016: Directions in Authenticated Ciphers*, 2016.

[IY09a] Tetsu Iwata and Kan Yasuda. BTM: A single-key, inverse-cipher-free mode for deterministic authenticated encryption. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *Selected Areas in Cryptography, 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers*, volume 5867 of *Lecture Notes in Computer Science*, pages 313–330. Springer, 2009.

[IY09b] Tetsu Iwata and Kan Yasuda. HBS: A single-key mode of operation for deterministic authenticated encryption. In Dunkelman [Dun09], pages 394–415.

[KR11] Ted Krovetz and Phillip Rogaway. The Software Performance of Authenticated-Encryption Modes. In Antoine Joux, editor, *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, volume 6733 of *Lecture Notes in Computer Science*, pages 306–327. Springer, 2011.

[LPTY16] Atul Luykx, Bart Preneel, Elmar Tischhauser, and Kan Yasuda. A MAC mode for lightweight block ciphers. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 43–59. Springer, 2016.

[Min14] Kazuhiko Minematsu. Parallelizable Rate-1 Authenticated Encryption from Pseudorandom Functions. In Nguyen and Oswald [NO14], pages 275–292.

[MM12] Seiichi Matsuda and Shiho Moriai. Lightweight cryptography for the cloud: Exploit the power of bitslice implementation. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, volume 7428 of *Lecture Notes in Computer Science*, pages 408–425. Springer, 2012.

[MPL+11] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In Kenneth G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 69–88. Springer, 2011.

[MRV15] Bart Mennink, Reza Reyhanitabar, and Damian Viz'ar. Security of full-state keyed sponge and duplex: Applications to authenticated encryption. In Iwata and Cheon [IC15], pages 465–489.

[Nan06] Mridul Nandi. A simple and unified method of proving indistinguishability. In Rana Barua and Tanja Lange, editors, *Progress in Cryptology - INDOCRYPT 2006, 7th International Conference on Cryptology in India, Kolkata, India, December 11-13, 2006, Proceedings*, volume 4329 of *Lecture Notes in Computer Science*, pages 317–334. Springer, 2006.

[Nan09] Mridul Nandi. Fast and secure cbc-type MAC algorithms. In Dunkelman [Dun09], pages 375–393.

[Nat80] National Institute of Standards and Technology. DES Modes of Operation. FIPS 81, December 1980.

[nis17] Lightweight Cryptography. https://www.nist.gov/programs-projects/lightweight-cryptography, 2017. Accessed: 2017-05-16.

[NO14] Phong Q. Nguyen and Elisabeth Oswald, editors. *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*. Springer, 2014.

[Pat91] Jacques Patarin. *Etude des générateurs de permutations pseudo-aléatoires basés sur le schéma du D. E. S.* PhD thesis, 1991.

[Pat08] Jacques Patarin. The "coefficients h" technique. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers*, 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 Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 33–63. Springer, 2016.

[RPLP08] Carsten Rolfes, Axel Poschmann, Gregor Leander, and Christof Paar. Ultra-Lightweight Implementations for Smart Devices - Security for 1000 Gate Equivalents. In Gilles Grimaud and François-Xavier Standaert, editors, *CARDIS*, volume 5189 of *Lecture Notes in Computer Science*, pages 89–103. Springer, 2008.

[RS06] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Vaudenay [Vau06], pages 373–390.

[Vau06] Serge Vaudenay, editor. *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*. Springer, 2006.

[WH16] Hongjun Wu and Tao Huang. The JAMBU Lightweight Authentication Encryption Mode (v2.1). CAESAR submissions, 2016. https://competitions.cr.yp.to/round3/jambuv21.pdf.

[ZWZW11] Liting Zhang, Wenling Wu, Lei Zhang, and Peng Wang. CBCR: CBC MAC with rotating transformations. *SCIENCE CHINA Information Sciences*, 54(11):2247–2255, 2011.

A Additional Result

Lemma 3. *Say that you have m sets of size n, which in turn are all subsets of a set of size N. Then the minimum size of the intersection of those sets is $mn - (m-1)N$.*

Proof. We prove it by induction on m. The case $m = 1$ holds because there are exactly n elements in the intersection. Assume it holds for $m-1$ sets, and we will prove the equation for m sets. Pick $m-1$ sets out of the m, and let α denote the minimum size of the intersection of those $m-1$ sets, i.e. $\alpha = (m-1)n - (m-2)N$. Then there are at most $N - \alpha$ elements in the big set which are not contained in the intersection of the $m-1$ sets. The remaining set must try to fit in those $N - \alpha$ elements, however there are at least $n - (N - \alpha)$ elements left in the remaining set which must end up in the intersection, giving us an intersection of size

$$n - (N - \alpha) = n - (N - (m-1)n + (m-2)N) = mn - (m-1)N\,. \tag{81}$$

□

B Transcript Graph Example

Say that $\vec{t}$ contains the following queries:

$$\begin{aligned}(\varepsilon,\varepsilon) \mapsto C_1, (\varepsilon, 10^{n-1}) \mapsto C_2, (10^{n-1}, \varepsilon) \mapsto C_3,\\ (10^{n-1}, 10^{n-1}) \mapsto C_4, (\varepsilon, 10^{n-1}10^{n-1}) \mapsto C_5,\\ (10^{n-1}10^{n-1}, \varepsilon) \mapsto C_6\,.\end{aligned} \tag{82}$$

Then split converts each of the above queries into sequences, which are prepended with $IV_{A,M}$:

$$(\varepsilon,\varepsilon) \to (0^n) \tag{83}$$

$$(\varepsilon, 10^{n-1}10^{n-1}) \to \left((010^{n-2}), (0, 10^{n-1}), (1, 10^{n-1})\right) \tag{84}$$

$$(10^{n-1}, \varepsilon) \to \left((100^{n-2}), (1, 10^{n-1})\right) \tag{85}$$

$$(10^{n-1}, 10^{n-1}) \to \left((110^{n-2}), (1, 10^{n-1}), (1, 10^{n-1})\right) \tag{86}$$

$$(\varepsilon, 10^{n-1}0^n) \to \left((010^{n-2}), (0, 10^{n-1}), (1, 0^n)\right) \tag{87}$$

$$(10^{n-1}10^{n-1}, \varepsilon) \to \left((100^{n-2}), (0, 10^{n-1}), (1, 10^{n-1})\right), \tag{88}$$

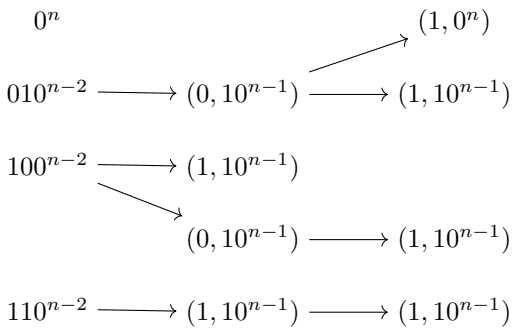


Figure 5: Graph induced by the sequences of Eq. (82) without the single-element sequences $((0, C_i[j]))$

The non-empty subsequences of the above sequences along with the single-element sequences $((0, C_i[j]))$ become the nodes of the induced graph $G_{\vec{t}}$:

$$(0^n), (010^{n-2}), (100^{n-2}), (110^{n-2}) \tag{89}$$

$$\Big((010^{n-2}), (0, 10^{n-1})\Big), \Big((010^{n-2}), (0, 10^{n-1}), (1, 10^{n-1})\Big) \tag{90}$$

$$\Big((100^{n-2}), (1, 10^{n-1})\Big) \tag{91}$$

$$\Big((110^{n-2}), (1, 10^{n-1})\Big), \Big((110^{n-2}), (1, 10^{n-1}), (1, 10^{n-1})\Big) \tag{92}$$

$$\Big((010^{n-2}), (0, 10^{n-1}), (1, 0^n)\Big) \tag{93}$$

$$\Big((100^{n-2}), (0, 10^{n-1})\Big), \Big((100^{n-2}), (0, 10^{n-1}), (1, 10^{n-1})\Big) \tag{94}$$

The labels of the nodes are the last elements of their sequence. Fig. 5 displays the resulting graph, where only the node labels are displayed and the $C_i[j]$ are not included.